

UNITED STATES PATENT AND TRADEMARK OFFICE

BEFORE THE PATENT TRIAL AND APPEAL BOARD

META PLATFORMS, INC.,
Petitioner

v.

TIR TECHNOLOGIES LIMITED,
Patent Owner.

IPR2026-00347

U.S. Patent No. 10,484,442

**PETITION FOR *INTER PARTES* REVIEW
OF U.S. PATENT NO. 10,484,442**

Mail Stop PATENT BOARD
Patent Trial and Appeal Board
U.S. Patent & Trademark Office
P.O. Box 1450
Alexandria, VA 22313-1450

TABLE OF CONTENTS

I.	FOUNDATIONS FOR STANDING.....	2
II.	IDENTIFICATION OF CHALLENGE	2
A.	Prior Art.....	2
B.	Grounds for Challenge	3
III.	'442 PATENT OVERVIEW	3
A.	Technology Background	3
B.	Overview	5
C.	Prosecution History	5
IV.	LEVEL OF ORDINARY SKILL	6
V.	CLAIM CONSTRUCTION	6
VI.	DETAILED EXPLANATION OF GROUNDS.....	7
A.	Ground 1A: Mattis in view of Hosur, with or without Sackstein.....	7
1.	Mattis.....	7
2.	Hosur	11
3.	Sackstein	14
4.	Motivation to Combine	14
5.	Claims 1/11	17
6.	Claims 2/12	32
7.	Claim 3/13	34
8.	Claims 4/14	34
9.	Claims 5/15	35
10.	Claims 6/16	35
11.	Claims 7/17	36
12.	Claims 8/18	37
13.	Claims 9/19	38
14.	Claims 10/20	39

B.	Ground 2A: Hosur in view of Mattis, with or without Sackstein.....	39
1.	Motivation to Combine	39
2.	Claims 1/11	41
3.	Claims 2/12	53
4.	Claim 3/13.....	55
5.	Claims 4/14	55
6.	Claims 5/15	56
7.	Claim 6/16.....	57
8.	Claims 7/17	58
9.	Claims 8/18	59
10.	Claims 9/19	59
11.	Claims 10/20	60
C.	Ground 3A: Mills in view of Mattis, with or without Sackstein.....	61
1.	Mills	61
2.	Motivation to Combine	62
3.	Claim 1/11	65
4.	Claims 2/12	74
5.	Claim 3/13.....	76
6.	Claims 4/14	76
7.	Claims 5/15	77
8.	Claims 6/16	77
9.	Claims 7/17	78
10.	Claims 8/18	79
11.	Claims 9/19	79
12.	Claims 10/20	80
VII.	CONCLUSION.....	81
VIII.	MANDATORY NOTICES	81

A. Real Party in Interest	81
B. Related Matters.....	81
C. Notice of Counsel and Service Information.....	82
D. Power of Attorney	83

EXHIBIT LIST

No.	Exhibit
1001	U.S. Patent No. 10,484,442
1002	File History of the '442 patent
1003	Declaration of Dr. Aviel D. Rubin
1004	Curriculum Vitae of Dr. Aviel D. Rubin
1005	U.S. Patent No. 6,289,358 (“Mattis”)
1006	U.S. Patent Application Pub. No. 2010/0070608 (“Hosur”)
1007	U.S. Patent Application Pub. No. 2009/0119322 (“Mills”)
1008	U.S. Patent Application Pub. No. 2009/0310668 (“Sackstein”)
1009	U.S. Patent No. 7,860,950 (“Menon”)
1010	Gourley, D., Totty, B., <i>HTTP: The Definitive Guide</i> , O’Reilly, (2002)
1011	Dilley, D., et al., <i>Globally Distributed Content Delivery</i> , IEEE Internet Computing (Sep. Oct. 2002)
1012	Buyya, R., Pathan, M., Vakali, A. (Eds.), <i>Content Delivery Networks</i> , Springer (2008)
1013	Dean, M., et al. (Eds.), <i>Web Information Systems Engineering – WISE 2005 Workshops</i> , Springer (2005)
1014	Plaintiff Tir Techs. Ltd.’s Responsive Claim Construction Brief, <i>Tir Techs. Ltd. v. Amazon.com Services LLC, et al.</i> , No. 25-dv-00865 (W.D. Tex. April 22, 2026)
1015	Schneier, B., <i>Applied Cryptography, Second Edition: Protocols, Algorithms, and Source Code in C</i> , John Wiley & Sons, Inc. (1996)

No.	Exhibit
1016	Ball, M., <i>A Quarter Century of Streaming Video</i> , https://www.thestreamingbook.com/chapters/a-quarter-century-of-streaming-video
1017	Ruether, T., <i>History of Streaming Media [Infographic]</i> , https://www.wowza.com/blog/history-of-streaming-media#h-1990-2000-streaming-debuts-as-a-proof-of-concept (Feb. 21, 2023)
1018	U.S. Patent Application Pub. No. 2002/0065926
1019	U.S. Patent Application Pub. No. 2011/0289533
1020	U.S. Patent Application Pub. No. 2004/0015986
1021	U.S. Patent Application Pub. No. 2004/0117455
1022	U.S. Patent Application Pub. No. 2002/0191696
1023	Shen, B., Lee, S-J., Basu, S., <i>Caching Strategies in Transcoding-enabled Proxy Systems for Streaming Media Distribution Networks</i> , HP Laboratories Palo Alto, HPL-2003-261 (2023)
1024	U.S. Patent Application Pub. No. 2003/0035481
1025	Deneke, T., <i>Scalable Distributed Video Transcoding Architecture</i> , Åbo Akademi Univ., Master of Science Thesis (Sep. 2011)
1026	U.S. Patent Application Pub. No. 2007/0198734
1027	U.S. Patent No. 7,941,453
1028	U.S. Patent Application Pub. No. 2004/0101138
1029	U.S. Patent Application Pub. No. 2007/0204064
1030	U.S. Patent Application Pub. No. 2011/0088076
1031	U.S. Patent Application Pub. No. 2004/0010613

No.	Exhibit
1032	Cacheda, F., Plachouras, V., Ounis, I., <i>A case study of distributed information retrieval architectures to index one terabyte of text</i> , Information Processing & Management, Vol. 41, Issue 5, pp. 1141-1161 (Sep. 2005)
1033	U.S. Patent Application Pub. No. 2007/0250900
1034	Complaint, <i>Tir Technologies Limited v. Meta Platforms, Inc.</i> , No. 8:25-cv-2345, Dkt. 1 (C.D. Cal. Oct. 16, 2025)

LISTING OF CHALLENGED CLAIMS

Reference	Limitation
Claim 1	
1[pre]	A computer-implemented method for streaming video, the method comprising:
1[a]	receiving a request to optimize an original source file from a client computing device, the request including a key specifying parameters to optimize the original source file;
1[b]	identifying, based on the key, a first portion of an optimized version of the original source file in a local cache of a video optimizer;
1[c]	generating a query, based on the key, to a database maintaining a plurality of reference keys corresponding to a plurality of optimized versions of source files stored on a plurality of other video optimizers, each reference key specifying parameters of an optimized version of a source file;
1[d]	receiving, from the database, a response indicating whether the key matches a reference key;
1[e]	receiving from the database, in response to a reference key matching the key, the address of another video optimizer storing a segment of the requested optimized version of the original source file;
1[f]	stitching the first portion of the optimized version of the original source file with the segment of the optimized version of the original source file stored on the other video optimizer; and
1[g]	streaming the stitched optimized version of the original source file for playback on the client computing device.
Claim 2	
2[a]	The method of claim 1, further comprising: storing a segment of the stitched optimized version of the original source file in a cache of the video optimizer; and

Reference	Limitation
2[b]	transmitting, to the database, instructions to update the database in response to storing the segment of the stitched optimized version of the original source file stored in the video optimizer.
Claim 3	
3	The method of claim 1, wherein the received response indicates that different segments of the optimized version of the same original source file are stored on a plurality of different video optimizers.
Claim 4	
4[a]	The method of claim 3, further comprising: transmitting, to the plurality of different video optimizers, a request for the segment of the optimized version of the original source file stored on each video optimizer; and
4[b]	stitching the different segments of the optimized version of the same original source file with the first portion of the original source file.
Claim 5	
5	The method of claim 1, wherein the key comprises a source content signature, the source content signature being a hash of the original source file.
Claim 6	
6	The method of claim 1, wherein the database comprises data entries identifying a plurality of video optimizers storing an optimized version of the original source file, the entries being modified through instructions transmitted to the database to update a location where the optimized version of the original source file is stored.

Reference	Limitation
Claim 7	
7[pre]	The method of claim 1, further comprising:
7[a]	determining a second segment of the optimized version of the original source file is not stored by the plurality of other video optimizers;
7[b]	transcoding, responsive to the determination, the second segment of the original source file;
7[c]	storing the transcoded second segment in the local cache of the video optimizer; and
7[d]	stitching the transcoded second segment with the stitched optimized version of the original source file.
Claim 8	
8	The method of claim 7, further comprising transmitting to the database, instructions to update the database with a reference key corresponding to the transcoded second segment of the optimized version of the source file stored at the video optimizer.
Claim 9	
9[a]	The method of claim 7, wherein transcoding the second segment of the original source file comprises: identifying a third segment of the optimized version of the original source file, subsequent to the second segment, that is stored by one of the plurality of other video optimizers; and
9[b]	transcoding the original source file from a start of the second segment to a start of the third segment.

Reference	Limitation
Claim 10	
10	The method of claim 1, wherein the first portion is maintained in the local cache of the video optimizer after one or more purges of the local cache.
Claim 11	
11[pre]	A system for streaming optimized video to a client device, comprising:
11[a]	a processor; and
11[b]	a non-transitory computer-readable storage medium storing executable computer program instructions that, when executed by the processor, causes the processor to:
11[c]	receive a request to optimize an original source file from a client computing device, the request including a key specifying parameters to optimize the original source file;
11[d]	identify, based on the key, a first portion of an optimized version of the original source file in a local cache of a video optimizer;
11[e]	generate a query, based on the key, to a database maintaining a plurality of reference keys corresponding to a plurality of optimized versions of source files stored on a plurality of other video optimizers, each reference key specifying parameters of an optimized version of a source file;
11[f]	receive, from the database, a response indicating whether the key matches a reference key;
11[g]	receive from the database, in response to a reference key matching the key, the address of another video optimizer storing a segment of the requested optimized version of the original source file;

Reference	Limitation
11[h]	stitch the first portion of the optimized version of the original source file with the segment of the optimized version of the original source file stored on the other video optimizer; and
11[i]	stream the stitched optimized version of the original source file for playback on the client computing device.
Claim 12	
12[a]	The system of claim 11, further comprising instructions that cause the processor to: store a segment of the stitched optimized version of the original source file in a cache of the video optimizer; and
12[b]	transmit, to the database, instructions to update the database in response to storing the segment of the stitched optimized version of the original source file stored in the video optimizer.
Claim 13	
13	The system of claim 11, wherein the received response indicates that different segments of the optimized version of the same original source file are stored on a plurality of different video optimizers.
Claim 14	
14[a]	The system of claim 13, further comprising instructions that cause the processor to: transmit, to the plurality of different video optimizers, a request for the segment of the optimized version of the original source file stored on each video optimizer; and
14[b]	stitch the different segments of the optimized version of the same original source file with the first portion of the original source file.
Claim 15	
15	The system of claim 11, wherein the key comprises a source content signature, the source content signature being a hash of the original source file.

Reference	Limitation
Claim 16	
16	The system of claim 11, wherein the database comprises data entries identifying a plurality of video optimizers storing an optimized version of the original source file, the entries being modified through instructions transmitted to the database to update a location where the optimized version of the original source file is stored.
Claim 17	
17[pre]	The system of claim 11, further comprising instructions that cause the processor to:
17[a]	determine a second segment of the optimized version of the original source file is not stored by the plurality of other video optimizers;
17[b]	transcode, responsive to the determination, the second segment of the original source file;
17[c]	store the transcoded second segment in the local cache of the video optimizer; and
17[d]	stitch the transcoded second segment with the stitched optimized version of the original source file.
Claim 18	
18	The system of claim 17, further comprising instructions that cause the processor to transmit to the database, instructions for updating the database with a reference key corresponding to the transcoded second segment of the optimized version of the source file stored at the video optimizer.
Claim 19	
19[a]	The system of claim 17, wherein transcoding the second segment of the original source file comprises: identifying a third segment of the optimized version of the original source file, subsequent to the

Reference	Limitation
	second segment, that is stored by one of the plurality of other video optimizers; and
19[b]	transcoding the original source file from a start of the second segment to a start of the third segment.
Claim 20	
20	The system of claim 11, wherein the first portion is maintained in the local cache of the video optimizer after one or more purges of the local cache.

By 2012, content-delivery networks (“CDNs”) provided content owners a way to serve websites, music, movies, and more to end users by storing and serving content on the owner’s behalf, rather than burdening their own servers. Because different devices with different capabilities would request access to the same content, CDNs needed ways to store, locate and serve different versions of content tailored for different devices without necessarily storing every possible version (which would require prohibitive amounts of memory). Further, even for versions the CDN stored, storing complete copies was an inefficient use of storage.

U.S. Patent No. 10,484,442 (the “’442 patent”) (Ex-1001) is directed to a CDN, but it did not offer anything new. Instead, it proposes two features that—as this Petition demonstrates—were long offered by prior-art CDNs. First, the ’442 patent proposes using a “key” to search a reference-key database to locate a specific “optimized” version of video content. Second, rather than store complete copies of different versions of the same video—requiring considerable storage—the ’442 patent proposes that the CDN stores limited “portions” of the different versions that are provided to the user immediately while the rest of the video is located or transcoded and streamed to the user.

Meta Platforms, Inc. (“Petitioner”) petitions for *inter partes* review (“IPR”) of claims 1-20 (“Challenged Claims”) of the ’442 patent.

I. GROUNDS FOR STANDING

Petitioner certifies the '442 patent is available for IPR, and Petitioner is not barred or estopped from requesting IPR on the grounds herein. Petitioner files this Petition within one year of service of a district court complaint by Patent Owner Tir Technologies Limited (hereafter, "PO"). Ex-1034.

II. IDENTIFICATION OF CHALLENGE

A. Prior Art

Petitioner respectfully requests cancellation of the Challenged Claims over:

Prior Art
U.S. 6,289,358 ("Mattis"); issued September 11, 2001; prior art under at least §102(b) (Ex-1005). ¹
U.S. 2010/0070608 ("Hosur"); published March 18, 2010; prior art under at least §102(b) (Ex-1006).
U.S. 2009/0119322 ("Mills"); published May 7, 2009; prior art under at least §102(b) (Ex-1007).
U.S. 2009/0310668 ("Sackstein"); published December 17, 2009; prior art under at least §102(b) (Ex-1008).

¹ All references to pre-AIA statutes. Because the '442 patent is a transitional application, even if post-AIA statutes apply, that would change only which provision under §102 each reference qualifies as prior art. *See Platinum Optics Tech. Inc. v. Viavi Sols. Inc.*, IPR2021-00631, 2022 WL 5056729, at *3 n.5 (PTAB Oct. 3, 2022).

B. Grounds for Challenge

Ground		Claims	Prior Art
1	§103	1-20	Mattis and Hosur, with or without Sackstein
2	§103	1-20	Hosur and Mattis, with or without Sackstein
3	§103	1-20	Mills and Mattis, with or without Sackstein

III. '442 PATENT OVERVIEW

A. Technology Background

To provide content like websites, music, or movies over the Internet, content owners typically maintain an “origin server” for storing and transmitting their content to end users upon request.

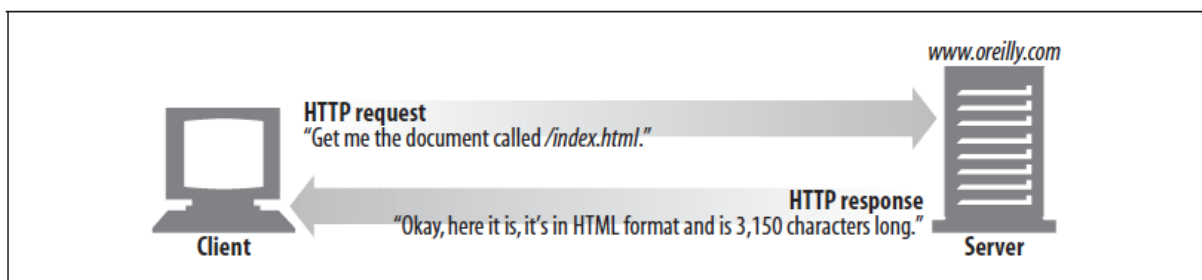


Figure 1-1. Web clients and servers

Ex-1010, Fig. 1-1

Ex-1003, ¶¶33-34, 36; Ex-1012, 5, 37. To avoid overloading content owners’ origin servers, however, other companies built large content-delivery networks (“CDNs”) of servers that would help distribute content on owners’ behalf. Ex-1003, ¶35; Ex-

1011, 50; Ex-1010, 163-64. CDNs “cache” (store) copies of content closer to end users for faster delivery without overburdening the owners’ origin servers. Ex-1003, ¶36; Ex-1012, 8-10.

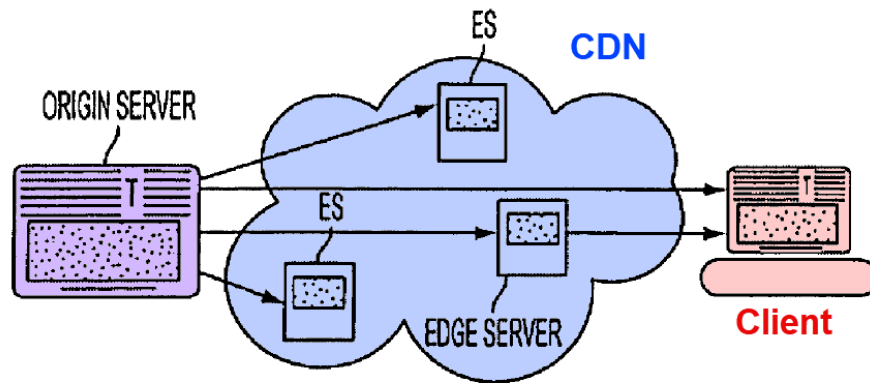


FIG. 1
(PRIOR ART)

Menon, Fig. 1, 3:47-65.²

Different end users often need different versions of the same content—such as different languages, file formats, or sizes—depending on their devices and preferences. Ex-1003, ¶37; Ex-1013, 43-52. CDNs therefore need ways to store, locate, and deliver different versions of the same content to different end users. Ex-1003, ¶37.

² All color annotations added.

B. Overview

The '442 patent describes a network that “optimizes” delivery of different versions of videos. Ex-1001, 1:43-56; Ex-1003, ¶¶38-41. When a client requests a video, the request includes a “key” representing: (1) the requested video, and (2) parameters associated with a specific version of the video. Ex-1001, 6:45-7:29; Ex-1003, ¶41.

The network compares the request’s key to a database of “reference keys” to determine whether a version of the video matching the request’s provided parameters exists. Ex-1001, 9:12-37, 12:30-54. A matching reference key means it does, but even if no matching reference key exists, the network uses the request’s parameters to generate (or “transcode”) a new version of the requested video matching the provided parameters. *Id.*, 9:12-37, 12:30-13:13; Ex-1003, ¶42.

The '442 patent also explains that storing complete copies of multiple versions of the same video is “an inefficient use of storage space.” Ex-1001, 8:22-42. Thus, only “portions” of different versions may be stored, which can be streamed immediately while the network locates or transcodes other portions. *Id.*; Ex-1003, ¶43.

C. Prosecution History

The '442 patent issued without any prior-art rejection. Ex-1002, 171-176, 207; Ex-1003, ¶¶44-45.

IV. LEVEL OF ORDINARY SKILL

A POSITA at the time of the alleged invention would have had at least a bachelor's degree in computer science, electrical engineering, or a related field, and at least two years of work or research experience in the field of networking and network-data management. Work experience can substitute for formal education and vice versa. Ex-1003, ¶¶46-48.

V. CLAIM CONSTRUCTION

No express constructions are necessary; Petitioners have applied the claim terms' ordinary meanings as understood by a POSITA.³ 37 C.F.R. §42.100(b); Ex-1003, ¶¶49-51.

In district-court litigation involving a third party, the third party identified several terms for construction: (1) several terms involving “optimization server”/“optimizer”/“to optimize” as requiring transcoding servers/transcoding; and (3) “stitch” as “combining into one file”. *See* Ex-1014. PO contended no construction is needed. *Id.* The district court has not issued any constructions, but the prior art taught the claims even if the third party's constructions were adopted.

³ Petitioners reserve the right to respond to PO/Board constructions. Petitioners reserve §112 arguments and express constructions consistent with this Petition.

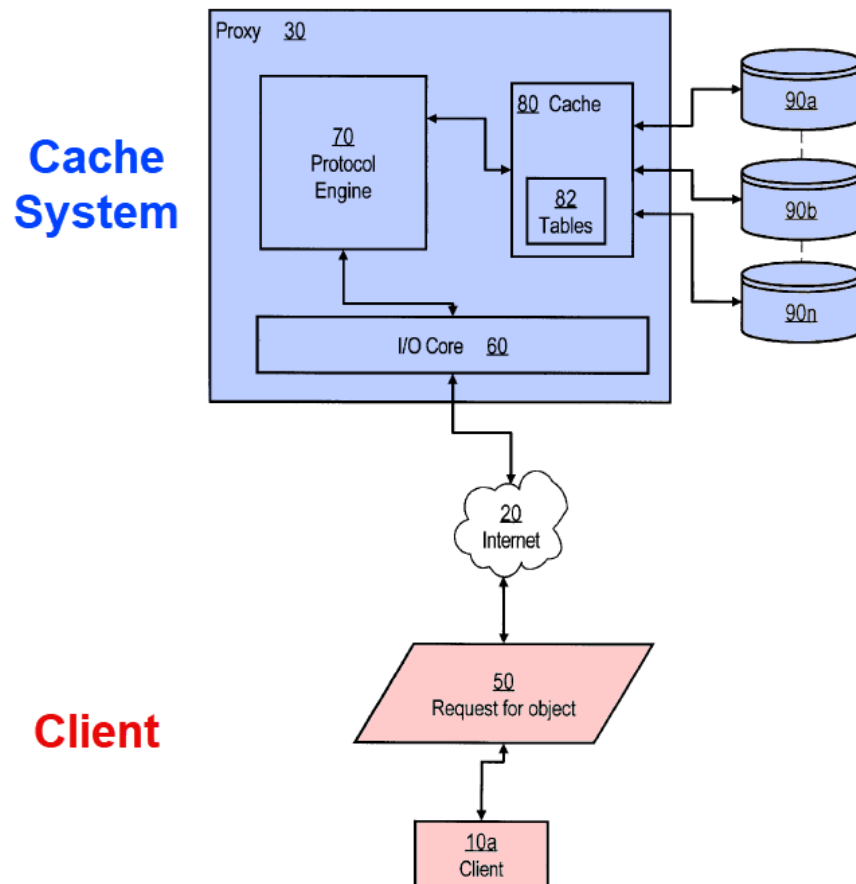
VI. DETAILED EXPLANATION OF GROUNDS

A. Ground 1A: Mattis in view of Hosur, with or without Sackstein

1. Mattis

Mattis discloses systems and methods “for caching and delivering” different versions of “multimedia objects” to clients. Mattis, Abstract, 2:16-18, 5:30-6:6; Ex-1003, ¶¶52-53. Mattis recognizes that clients often need different versions of the same objects—such as different languages or formats—and there is a need to search caches quickly to fulfill requests. Mattis, 1:53-2:52, 5:33-41, 14:16-20, 14:34-36, 14:44-48; Ex-1003, ¶53.

FIG. 2

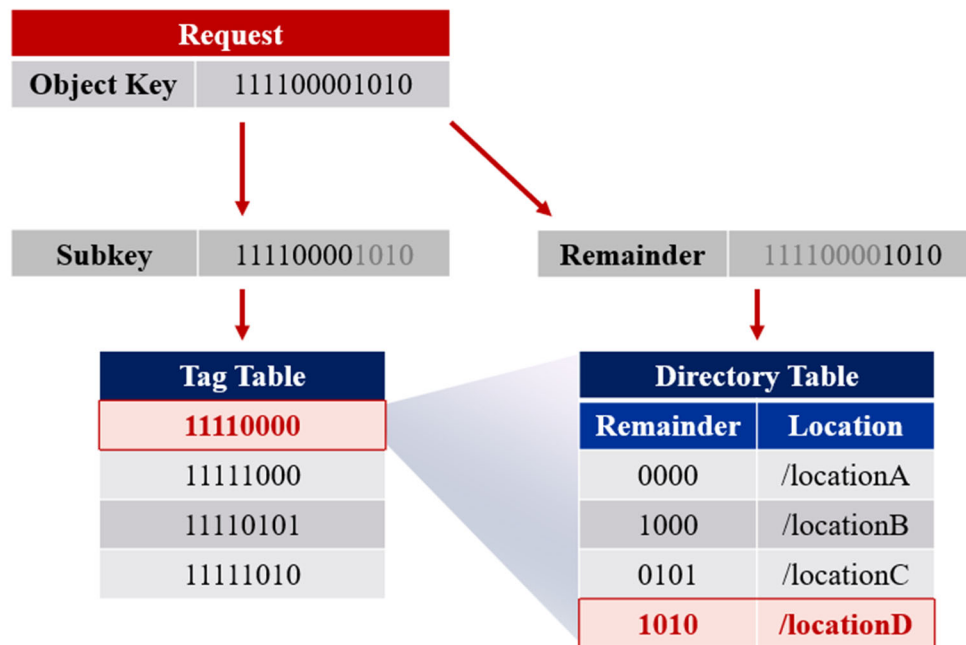


Mattis, Fig. 2.

To do so, Mattis includes a “key” in client requests. Mattis, 8:12-15. “There are two types of keys, called ‘name keys’ and ‘object keys.’” *Id.*, 8:15-16. An “object key” represents a specific version of a specific object (e.g., “petition.doc”). *Id.*, 8:27-45. But because clients may not know the exact versions available (*id.*, 10:10-22), a “name key” represents: (1) the name of a specific object (e.g., “petition”); and (2) parameters associated with a specific version of the object (e.g., browser, language, format). *Id.*, 8:13-26, 10:10-22, 14:9-25; Ex-1003, ¶¶55-57.

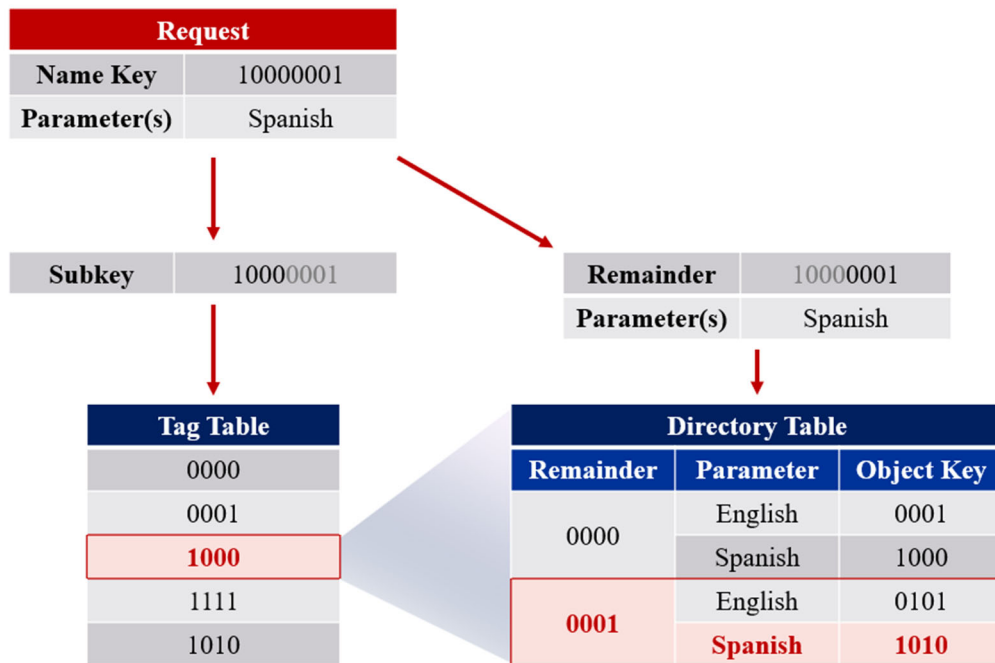
When a request is received, its key is compared to directories of reference keys (e.g., “Tag” and “Directory” Tables) to find a match. Mattis, 11:36-12:49. Each directory contains part of the key for faster searching. *Id.*, 9:35-45. Specifically, the Tag Table searches for a first part of the key to determine if the requested object is *not* being stored and the search can stop. *Id.* But if a match exists in the Tag Table, the rest of the key is searched in the Directory Table. *Id.*, 9:46-57, 11:36-12:49; Ex-1003, ¶58.

For object keys, a matching entry in the Directory Table identifies the location of the specific version of the object (Mattis, 11:58-67), as shown below.



Ex-1003, ¶58; *see also* Mattis, Fig. 4A

For name keys, a matching key in the Directory Table identifies: (1) each version of the object, (2) parameters associated with each version, and (3) each version's object key. Mattis, 11:67-12:4, 14:37-55. The parameters in the request (e.g., browser, language, format) are used to select the best-matching version in the Table, and the object key then points to the location of that version in the cache. *Id.*, 14:8-15:51.



Ex-1003, ¶59; *see also* Mattis, Fig. 4A

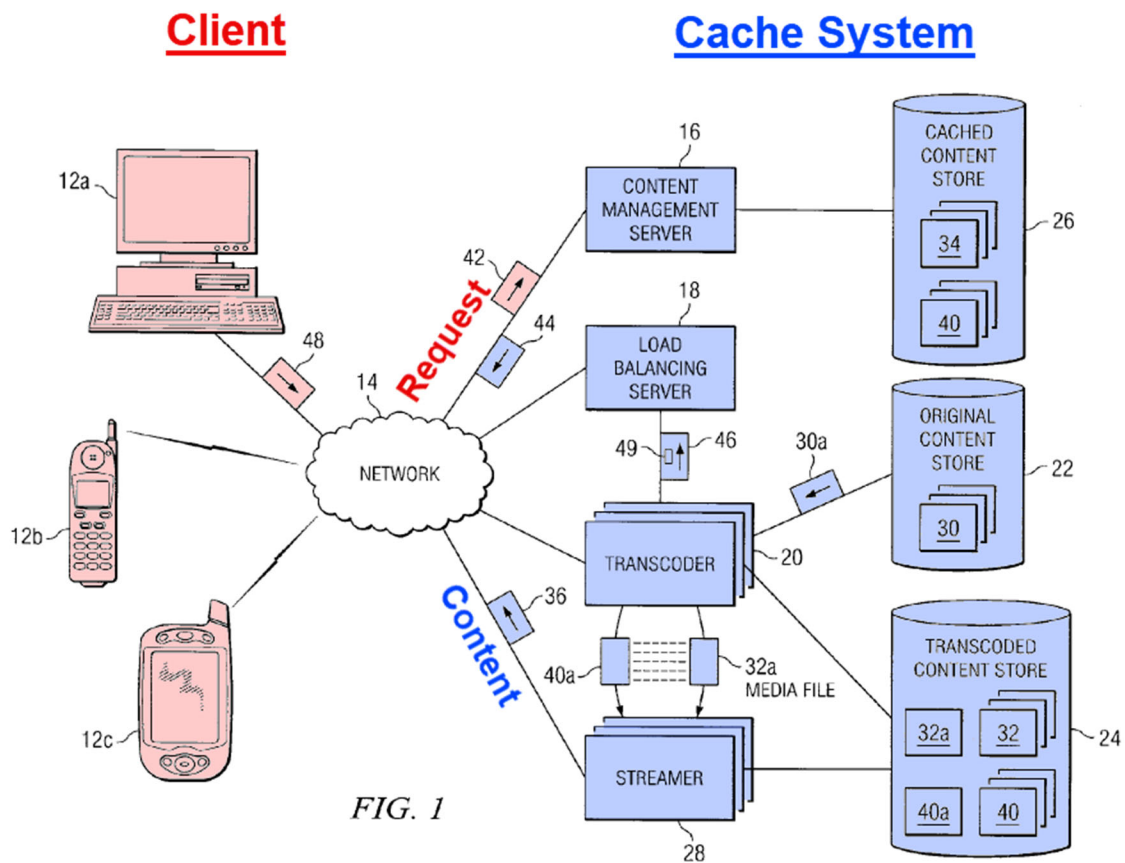
Thus, all searches eventually use an object key to locate a specific version of the requested object. Ex-1003, ¶59.

Finally, Mattis discloses that large objects are split and stored as smaller “fragments.” Mattis, 19:19-20:53. An object key points to the fragments, and

fragments of the same object are “chained” so they may be collected and combined when requested. *Id.*; Ex-1003, ¶60.

2. Hosur

Hosur discloses systems and methods “for delivering media content” by “receiving a request identifying a requested media file.” Hosur, Abstract; Ex-1003, ¶61-66. Hosur and Mattis are similar, with a few differences. Ex-1003, ¶67.

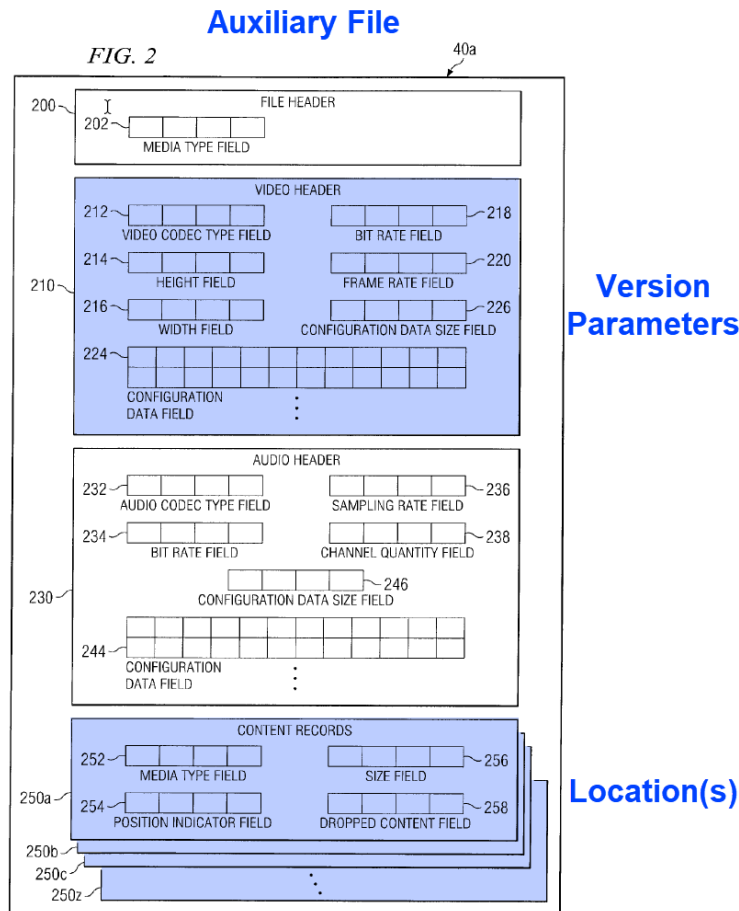


Hosur, Fig. 1.

First, whereas Mattis stores “multimedia objects” such as “documents and images” (*see, e.g.*, Mattis, 1:37-39, 2:16-18, 5:30-41), Hosur specifically concerns audio and video (Hosur, ¶¶2, 16-17, 18, 32). Ex-1003, ¶67.

Second, whereas Mattis stores different existing versions of objects (Mattis, 14:26-55), Hosur can create new versions upon request using “transcoders” (Hosur, ¶¶3, 5, 18, 24, 26). Ex-1003, ¶67.

Finally, whereas Mattis uses keys and databases to locate a specific version of requested content (Mattis, 8:8-45), Hosur uses an “auxiliary file” that specifies the parameters associated with, and location(s) of, each version in the cache (Hosur, ¶¶5-6, 27, 46-48). Ex-1003, ¶67.



Hosur, Fig. 2.

In Hosur, when a client requests a video, the request identifies the video and parameters for a specific version of the video (e.g., “the codec, bit-rate, and/or other parameters”). *Id.*, ¶39. This is like Mattis’s “name key.” Ex-1003, ¶67. If the cache already has a version with matching parameters, it is provided to the client. Hosur, ¶¶40, 95, Fig. 5. Otherwise, a transcoder converts the video into the requested format. *Id.*, ¶¶5-6, 27, 41-42, 46-48, 96, Fig. 5; Ex-1003, ¶¶64-65.

Because video transcoding takes time, transcoders store and/or transcode “portions” of requested video, which may be streamed to the client while other portions are being transcoded, and each portion’s location is tracked using the auxiliary file. Hosur, ¶¶3, 44-49; Ex-1003, ¶65.

3. Sackstein

Sackstein discloses a system and method for concurrently transcoding portions of video to adapt to network conditions (e.g., reducing a video’s size when bandwidth is limited). Sackstein, Abstract; Ex-1003, ¶¶71-76. Specifically, Sackstein provides a plurality of transcoders capable of concurrently transcoding multiple streams of video. Sackstein, ¶28. Before transcoding, Sackstein checks whether a specific version of video (or portions) already exists. *Id.*, ¶¶56-58; Ex-1003, ¶¶73-75. If not, portions of the video are sent to one or more transcoders to create a new version, which are spliced together and streamed to the client. Sackstein, ¶¶40, 58; Ex-1003, ¶75. Because Sackstein uses multiple transcoders, different portions of a video may be stored in different caches. Sackstein, ¶47; Ex-1003, ¶76. The system therefore searches both local and neighboring caches to determine which portions have already been converted. Ex-1003, ¶76.

4. Motivation to Combine

Mattis, Hosur, and Sackstein are all analogous art to the ’442 patent and each other, because each is directed to networks for storing and delivering different

versions of content to clients, and a POSITA would have been motivated to combine them. Ex-1003, ¶¶77-79.

Although Mattis does not list video in its exemplary “multimedia objects” (Mattis, 1:37-39, 2:16-18, 5:30-41), video was a conventional multimedia form. Ex-1003, ¶¶80-82 (citing Ex-1018, ¶2; Ex-1019, ¶64; Ex-1020, ¶42). At Mattis’s time (1998), streaming video over the Internet was limited due to bandwidth constraints. Ex-1003, ¶80; Ex-1016. But by 2012, streaming video was prevalent, and a POSITA would have been motivated to include video in Mattis’s system as another form of multimedia, confirmed by Hosur. Ex-1003, ¶81. A POSITA would have had a reasonable expectation of success because Mattis envisions handling “any discrete element of information that is delivered from a server” (Mattis, 7:23-27)—of which video is one—and handling video data is no different than handling other multimedia data Mattis discloses, only with typically bigger files. Ex-1003, ¶83.

When incorporating video into Mattis’s system, a POSITA would have also been motivated to incorporate transcoders, as taught by Hosur, to generate new versions of video upon request. *Id.*, ¶84. Cache memory is limited, and video is memory-intensive. *Id.* Rather than store copies of every possible version of the same video, storing some (e.g., popular) versions, or smaller portions of each version, and transcoding the rest upon request would “reduce[] storage requirements,” while “servic[ing] a greater number and variety of users and eliminate the time and expense

of pre-transcoding all content.” Hosur, ¶60; *id.*, ¶7; Ex-1003, ¶85 (citing Ex-1022, ¶44; Ex-1023, 2-4). A POSITA would have had a reasonable expectation of success in making this modification because Mattis envisioned “software for dynamically generating...resources in response to requests” (Mattis, 1:39-41), and transcoders were conventional for dynamically generating video (Hosur, ¶15; Ex-1024, ¶10, Fig. 1; Ex-1022, ¶44). Ex-1003, ¶86. Further, storing, transcoding, and delivering video portions would have been straightforward because Mattis already taught storing “fragments” (portions) of large objects (Mattis, 19:19-20:53), and Hosur taught transcoding individual video “portions” (Hosur, ¶¶3, 44-49). Ex-1003, ¶86.

Finally, Hosur envisions multiple transcoders (Hosur, ¶15)—each with transcoded-content stores (*id.*, ¶¶31, 34)—operating “concurrently” (*id.*, ¶78). Although Hosur primarily describes transcoding and storing all portions of a specific version of video in a single transcoder (*id.*, ¶¶42-51), a POSITA would have understood that, to complete transcoding faster, Hosur (and thus Mattis-Hosur combined) contemplates having different transcoders transcode (and store) different portions before combining them for streaming. Ex-1003, ¶¶87-90.

Regardless, Sackstein describes a system/method for segmenting and storing multimedia like Mattis (Sackstein, ¶¶30-35), and transcoding video segments as needed like Hosur (*id.*, ¶¶39-40, 56-58). Ex-1003, ¶91. But Sackstein expressly utilizes multiple transcoders concurrently (Sackstein, ¶¶22, 28-29), and searches for

video segments in multiple caches (*id.*, ¶47). Ex-1003, ¶91. A POSITA would have been motivated to expressly enable Mattis-Hosur to concurrently transcode and store different portions of a video in different transcoders to complete transcoding faster, and also offer dynamic transcoding of different portions to adapt to network conditions, like Sackstein. Ex-1003, ¶92. A POSITA would have had a reasonable expectation of success in making this modification because Mattis-Hosur combined already taught multiple transcoders (Hosur) and storing and locating fragments/portions of objects in different memory stores (Mattis). Ex-1003, ¶93. The only possible change would be to expressly enable different transcoders to transcode and store different portions of the same video. *Id.*

5. Claims 1/11

a. 1[pre]/11[pre]

If limiting, Mattis-Hosur teaches/suggests 1[pre]/11[pre]. Ex-1003, ¶¶94-98.

Mattis discloses “[a] method and apparatus for caching information objects” (Mattis, 7:4-12) and delivering them “customized for the requesting client” (*a system/computer-implemented method for streaming...[to a client device]*) (Mattis, Abstract). *See also id.*, 1:4-2:14, 7:4-12, 36:39-46 (“the methods described herein

are carried out using a general-purpose programmable digital computer system”); 39:12-17 (streaming), Figs. 1-2, 11; Ex-1003, ¶96.⁴

“[I]nformation objects” are multimedia including “Web pages or documents, graphic[al] images, files, text documents...or other elements stored on a server.” Mattis, 7:15-29, 5:30-41. Each object “may be associated with a plurality of versions,” and “the appropriate version...is selected” (*optimized*). *Id.*, 5:30-57; Ex-1003, ¶97.

Although Mattis does not expressly list “video,” Hosur taught a system like Mattis’s that cached different versions of video, and selected a specific version to stream to the client (*streaming [optimized] video*). *See, e.g.*, Hosur, ¶¶15, 25-28, 39-44 (transcoder/streamer operations), 32-33 (video storage), 12, Fig. 4 (streamer). It would have been obvious to use Mattis’s system to stream video, consistent with Hosur. §VI.A.4 (motivation); Ex-1003, ¶98.

b. 11[a]

Mattis discloses/suggests 11[a]. Ex-1003, ¶¶99-100.

Mattis discloses “computer system 1100 upon which...the invention may be implemented” includes “*a processor*.” Mattis, 36:56-37:35; Ex-1003, ¶100.

⁴ Italics refers to claim language.

c. 11[b]

Mattis discloses/suggests 11[b]. Ex-1003, ¶¶101-02.

Mattis discloses “[c]omputer system 1100 also includes a main memory 1106...for storing information and instructions to be executed by processor 1104” (*non-transitory computer-readable storage medium storing executable computer program instructions that when executed by the processor, cause the processor to*). Mattis, 36:56-38:6; Ex-1003, ¶102. The instructions would cause the processor to perform each of the functions addressed below, either alone or in combination with Hosur and/or Sackstein. Ex-1003, ¶102.

d. 1[a]/11[c]

Mattis, alone or combined with Hosur, teaches/suggests 1[a]/11[c]. Ex-1003, ¶¶103-113.

Mattis discloses “[a] client 10a directs a request 50 for an object” and “[t]he incoming request 50 arrives at an input/output (I/O) core 60” (*receiv[ing] a request...from a client computing device*). Mattis, 7:19-65; *id.*, 5:30-57, 1:16-22 (noting clients are computers). The request is used to select a specific version of an object. *Id.*, Abstract, 5:30-57, 9:16-21, 10:10-22; Ex-1003, ¶104.

Client requests include an object or name key (*a key*). Mattis, 8:16-17, 10:10-22; Ex-1003, ¶¶105, 108-09 (citing Ex-1026, ¶15). An object key is a “unique fingerprint or compressed representation of the contents of the object,” preferably

generated by running the object through a “hash function.” Mattis, 8:26-45. Mattis’s “object key” is a *key specifying parameters to optimize the original source file* because it identifies a specific version of a specific source file created using specific parameters. Ex-1003, ¶106.

A “name key” represents an object’s name. Mattis, 10:10-21. When a request includes a “name key,” the request also includes parameters about the client or client device (e.g., browser, language, or format, *id.*, 14:9-25)—to identify a specific version of the object requested by name (*id.*, 14:26-15:51, 28:39-30:14, Figs. 3B, 4A, 9A, C, E). Ex-1003, ¶107. These parameters may be incorporated into the name key (*key specifying parameters to optimize the original source file*). Mattis, 14:56-65.

Thus, Mattis discloses a request “*to optimize an original source file*” is a request for a specific version of an object, consistent with how the ’442 patent describes a request “*to optimize.*” Ex-1001, 11:15-12:54 (optimizing by “obtain[ing] the optimized version...which may be...retrieved from a local cache”), Fig. 3A; Ex-1003, ¶110.

However, if “a request *to optimize*” requires requesting generation of a new version of an object, Hosur taught requests that initiated transcoding of a new version of a video (*to optimize*) when the requested version does not already exist (Hosur, ¶¶39-44, 91-98, Fig. 5 (transcoder operation based on request)), just like the ’442

patent (Ex-1001, 12:30-54); Ex-1003, ¶¶111-12. It would have been obvious to incorporate transcoders like Hosur's into Mattis's system to generate new versions of video upon request. §VI.A.4 (motivation); Ex-1003, ¶113.

e. 1[b]/11[d]

Mattis-Hosur teaches/suggests 1[b]/11[d]. Ex-1003, ¶¶114-18.

Mattis discloses that, upon receiving a request, a specific version of an object is located using the object key, or name key and associated parameters. Mattis, 8:6-45 (object key), 10:9-34 (name key), 14:9-15:51 (version parameters), 28:39-30:14 (operation), Figs. 3B, 4A, 9A, C, E; Ex-1003, ¶115.

Mattis discloses that objects may be stored as “fragments” (portions). Mattis, 17:1-18:21. The name/object key is used to locate fragments—including a first fragment—of a specific version of an object (*identify[ing], based on the key, a first portion of an optimized version of the original source file*). *Id.*, 19:19-64; Ex-1003, ¶116.

Although Mattis does not store video in a “video optimizer” as claimed, Hosur teaches storing portions of video in a transcoder's local “transcoded-content store” (*a local cache of a video optimizer*) (Hosur, ¶¶34, 44), and delivering a first portion while transcoding other portions (*id.*, 46-49). Ex-1003, ¶117. When implementing Mattis to transcode and stream video (§VI.A.5.d (1[a]/11[c])), a first portion (fragment) of requested video would be stored and streamed from transcoders to

deliver the specific version quickly while the rest transcodes, consistent with Hosur. §VI.A.4 (motivation); Ex-1003, ¶118. Thus, the combination teaches the key is used to identify a first portion *in a local cache of a video optimizer*. Ex-1003, ¶¶117-18.

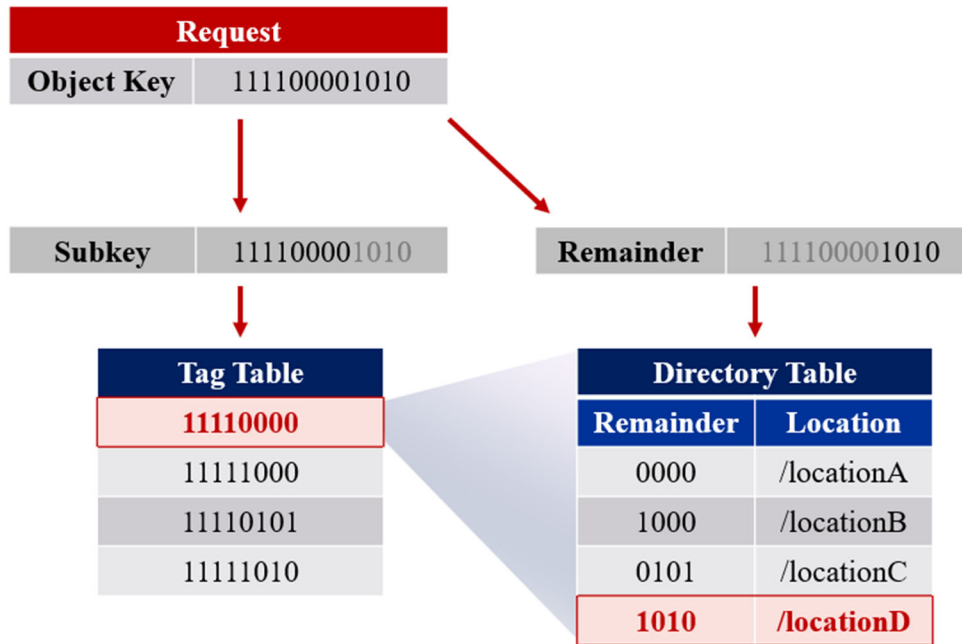
f. 1[c]/11[e]

Mattis-Hosur teaches or suggests 1[c]/11[e]. Ex-1003, ¶¶119-28.

Mattis discloses directories—including Tag and Directory Tables (*database*)—are searched using the request’s object or name keys (*generat[ing] a query, based on the key, to a database*). Mattis, 10:49-53; Ex-1003, ¶¶120-22. The directories store reference object and name keys associated with different versions of objects stored in the cache (*maintaining a plurality of reference keys corresponding to a plurality of optimized versions of source files*). Mattis, 8:12-18 (“Keys are used to index into directories that point to the location of objects on disk”), 8:42-45 (“Since the keys...serve as indexes into tables of the cache 80, the cache is referred to as a content-indexed cache. Given a content fingerprint key, the content can easily be found”), 12:9-49 (describing a search); Ex-1001, 2:61-63 (’442 patent explaining “[t]he cache database maintains entries corresponding to each optimized video locally cached at a given optimizer”); Ex-1003, ¶¶120-21 (citing Ex-1027, 6:55-60; Ex-1028, ¶126; Ex-1029, ¶67).

For object keys, a matching key in the Tag/Directory Tables identifies the location of a specific version of an object. Mattis, 11:58-67; *see also id.*, 8:40-45,

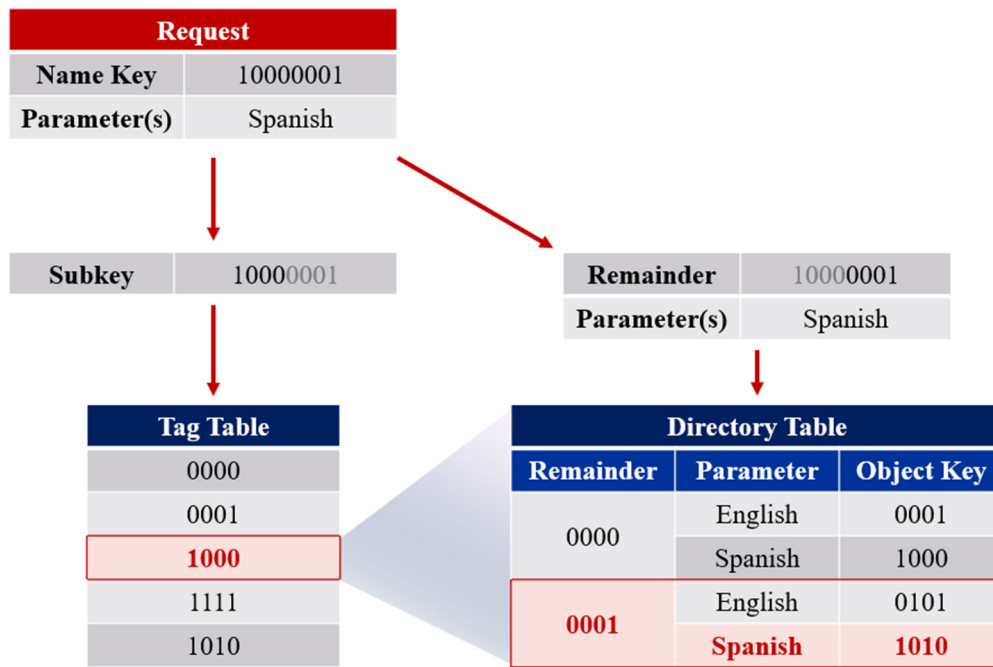
10:36-11:63 (search process, generally), 28:39-30:14 (example operation), Fig. 4A;
Ex-1003, ¶123.



Ex-1003, ¶123

Object keys in the Tag/Directory Tables thus *specify parameters of an optimized version of a source file* because the keys identify specific versions of objects created using specific parameters. *Id.* Objects are stored in one of a plurality of “storage devices 90a-90n” (Mattis, 8:7-9, 11:53-57), and object keys in the Tag/Directory Tables point to “a disk location” of the object; thus the directories contain *a plurality of reference keys corresponding to a plurality of optimized versions of source files stored on a plurality of other...video optimizers.* *Id.*, 11:22-67; Ex-1003, ¶124.

For name keys, a matching name key in the Tag/Directory Tables identifies all versions of the object (a “Vector of Alternates”). Mattis, 11:67-12:8; Ex-1003, ¶125. The “Vector of Alternates...stores a plurality of alternate records...that describe[] an alternative version of the requested object,” such as versions for particular applications, languages, or formats (*reference keys...specifying parameters of an optimized version of a source file*). Mattis, 14:37-48; Ex-1003, ¶125. The request’s parameters create a “vector” that points to a specific version in the alternate records. Mattis, 14:9-36; Ex-1003, ¶126. “The alternate records also each store a full object key” (Mattis, 14:51-55), which specifies the location of each version of the object in one of the plurality of storage devices (*a plurality of reference keys corresponding to a plurality of optimized versions of source files stored on a plurality of other...video optimizers*) (*id.*, 8:7-9, 11:22-67).



Ex-1003, ¶126

Although Mattis does not store different versions of video in video optimizers, Hosur teaches transcoders each with a “[t]ranscoded content store 24 [that] stores transcoded media files,” including “partial files.” Hosur, ¶33; *id.*, ¶¶31 (“content stores may each represent a plurality of devices and may be distributed across multiple locations”). Ex-1003, ¶127. When implementing Mattis to transcode and stream video (§VI.A.5.d (1[a]/11[c])), Mattis’s Tag/Directory Tables would point to the disk locations of different transcoded-content stores storing different portions of videos (*a plurality of optimized versions of source files stored on a plurality of other video optimizers*), consistent with Hosur. §VI.A.4 (motivation); Ex-1003, ¶128.

g. 1[d]/11[f]

Mattis discloses/suggests 1[d]/11[f]. Ex-1003, ¶¶129-31.

Mattis discloses that a request's name or object key (*the key*) is compared to keys in the Tag/Directory Tables (*reference keys*) to determine whether there is a match. Mattis, 11:36-12:49 (“The search seeks a match of the tag subkey 59 to one [of] the entries. If a match is found...the cache 80 proceeds to seek a hit in the Directory Table”); *id.*, 9:35-57, 10:54-11:4, 28:37-36:36 (describing a search); Ex-1003, ¶130. The key comparison returns an instruction (e.g., bit-flag/update (Mattis, 29:11-40), read/copy (29:55-62, 32:60-66), success/fail indication (30:4-19), error message (33:25-63, 34:25-60)) to, e.g., the cache and/or a protocol engine depending on whether there is a match and other conditions (*receiving, from the database, a response indicating whether the key matches a reference key*). See also *id.*, 31:5-18 (protocol engine); Ex-1003, ¶131.

h. 1[e]/11[g]

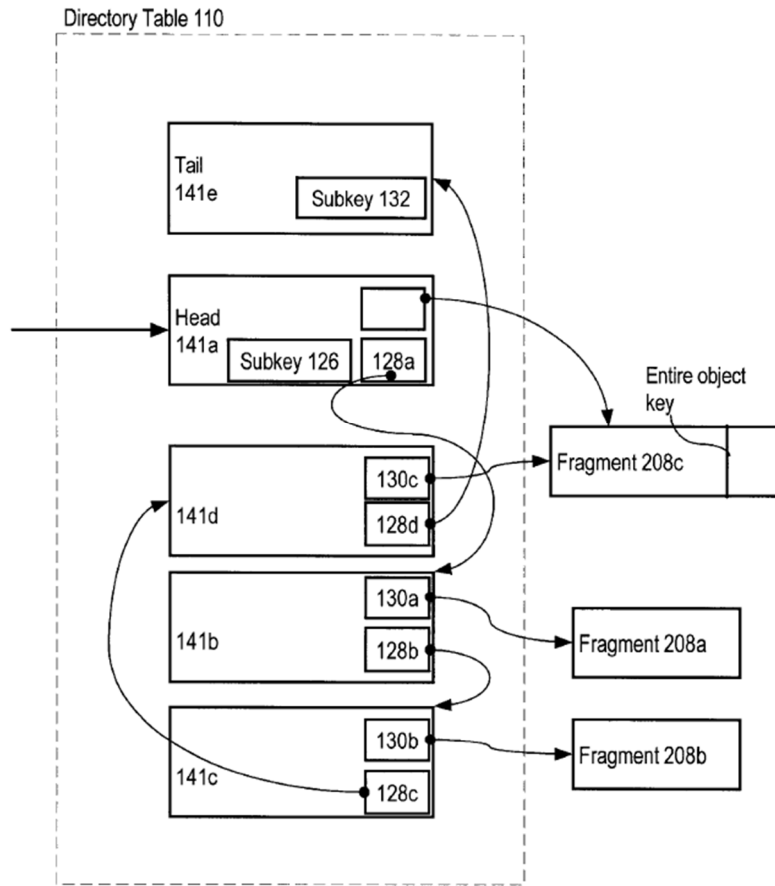
Mattis-Hosur, with or without Sackstein, teaches/suggests 1[e]/11[g]. Ex-1003, ¶¶132-36.

Mattis discloses that when requests include object keys, a matching key in the Tag/Directory Tables points to the location of the requested object. Mattis, 11:58-12:4; §VI.A.5.f (1[c]/11[e]). When requests include a name key, a matching key in the Tag/Directory Tables provides the “alternate records” for the different versions

of the requested object with their parameters, which are used to identify the object key—and thus the location—of the best-matching version. Mattis, 14:26-36, 15:13-31; §VI.A.5.f (1[c]/11[e]). Mattis thus discloses *receiv[ing] from the database, in response to a reference key matching the key, [an] address* of the optimized version of the original source file. Ex-1003, ¶133.

Mattis also discloses that “[t]he data for an entire stored object...may be stored within multiple fragments.” Mattis, 17:59-66. An object-key search in the Directory Table provides a “[b]lock [that] is the head of a chain of blocks that identify successive fragments that contain the object.” *Id.*, 19:39-47.

FIG. 5



Accordingly, by matching the request's object key to one in the Tag/Directory Tables, Mattis identifies the locations of different "fragments" (*segments*) of the requested object. Ex-1003, ¶134.

Although Mattis does not expressly disclose storing and retrieving different portions of a video from different video optimizers, Hosur at least suggests this. §VI.A.4 (motivation) (citing Hosur, ¶¶15, 31, 34, 78). When combining Mattis-Hosur, the object/name key in Mattis's Tag/Directory Tables would identify the address of the different transcoded-content stores (*video optimizers*) storing different

fragments/portions (*storing a segment*) of the requested version of video, consistent with Hosur (*receiv[ing] from the database, in response to a reference key matching the key, [an] address of another video optimizer storing a segment of the requested optimized version of the original source file*). §VI.A.4 (motivation); Ex-1003, ¶135.

Regardless, Sackstein expressly teaches transcoding and storing different portions of a video in different transcoders. §VI.A.4 (motivation) (citing Sackstein, ¶¶22, 28-35, 39-40, 47, 56-58). When implementing Mattis-Hosur to transcode and store different portions of a video in different transcoders, consistent with Sackstein, the *address* received from Mattis’s Tag/Directory Tables would be the address of *another video optimizer storing a segment of the requested optimized version of the original source file*. *Id.*; Ex-1003, ¶136.

i. 1[f]/11[h]

Mattis-Hosur, with or without Sackstein, teaches/suggests 1[f]/11[h]. Ex-1003, ¶¶137-40.

Mattis discloses that “[t]he object store needs a mechanism to chain fragments together” for retrieval (Mattis, 19:65-66), and “large objects can be efficiently pipelined from the object store” for streaming, (*id.*, 39:15-17). Queuing chained fragments in a pipeline for streaming constitutes *stitching* consistent with the ’442 patent. Ex-1001, 17:18-41; Ex-1003, ¶138.

Although Mattis does not expressly disclose chaining fragments of video from different video optimizers, Hosur suggests storing different transcoded portions of a video in different transcoded-content stores. §VI.A.4 (motivation) (citing Hosur, ¶¶15, 31, 34, 78). Hosur also discloses combining portions into a dynamic “transcoded media file” that “allow[s] streaming of content...while transcoded media file 32a...[is] still being written.” Hosur, ¶48; *id.*, ¶¶46-47, 53-56; Ex-1003, ¶139. When implementing Mattis to transcode and store different portions of a video on different transcoders consistent with Hosur, a streaming element would *stitch* a first fragment of a video (*first portion of the optimized...source file*) with subsequent portions stored in different transcoded-content stores (*stitch[ing]...with the segment of the optimized...source file stored on the other video optimizer*). Ex-1003, ¶139. This would be done by either chaining fragments/portions in a pipeline like Mattis, or by compiling them into a dynamic “transcoded media file” like Hosur, as these were two known alternative ways to combine portions of files together for streaming. *Id.*

Regardless, Sackstein expressly teaches storing different video portions in different transcoders, searching different transcoder caches for portions of a video, and “splicing” them together for streaming to the client. §VI.A.4 (motivation) (citing Sackstein, ¶¶22, 28-35, 39-40, 47, 56-58). When implementing Mattis to transcode and store different portions of a video on different transcoders consistent with Hosur

and/or Sackstein (§VI.A.5.h (1[e]/11[g])), the portions would be collected from the different transcoders and chained/combined/spliced together for streaming to the client. §VI.A.4 (motivation); Ex-1003, ¶140.

j. 1[g]/11[i]

Mattis-Hosur, with or without Sackstein, teaches/suggests 1[g]/11[i]. Ex-1003, ¶¶141-43.

Mattis transmits chained fragments of a specific version of an object to the client. §VI.A.5.i (1[f]/11[h]). “The cache uses streaming I/O, so large objects can be efficiently pipelined from the object store to slow clients, without staging the entire object into memory” (*streaming*). Mattis, 39:12-17; Ex-1003, ¶142.

Although Mattis does not disclose streaming video for playback on client devices, Hosur and Sackstein each teach streaming combined/spliced video for client playback (*stream[ing] the stitched optimized version...for playback on the client computing device*). §§VI.A.5.i (1[f]/11[h]); Hosur, ¶¶48, 53-54, 57; Sackstein, ¶¶33, 41. When implementing Mattis to store and stream video consistent with Hosur, chained/combined/spliced fragments of video would be streamed *for playback on the client computing device*, consistent with Hosur and/or Sackstein, because video is intended to be played back. §VI.A.4 (motivation); Ex-1003, ¶143.

6. Claims 2/12

a. 2[a]/12[a]

Mattis-Hosur, with or without Sackstein, teaches/suggests 2[a]/12[a]. Ex-1003, ¶¶144-46.

Mattis discloses pipelining fragments for streaming (Mattis, 39:15-17), meaning fragments would be (at least temporarily) copied into the cache handling streaming. Ex-1003, ¶145. Mattis separately discloses that fragments are reorganized/rewritten into new memory areas for efficiency/clean-up. Mattis, 21:65-22:9, 23:33-47; Ex-1003, ¶145. Accordingly, Mattis discloses or suggests that fragments/objects initially stored in different memory can be copied and/or rewritten into a single memory. Ex-1003, ¶145. Additionally, Hosur discloses combining portions into a single “transcoded media file” after transcoding. §VI.A.5.h (1[e]/11[g]); Ex-1003, ¶145.

When implementing Mattis to transcode and store fragments/portions of a video on different transcoders consistent with Hosur and/or Sackstein (§VI.A.5.h (1[e]/11[g])), the fragments/portions from different transcoders would be collected in a single transcoder—either copied/reorganized consistent with Mattis, or combined into a single “transcoded media file” consistent with Hosur—(*storing a segment of the stitched-optimized version...in the cache of the video optimizer*) for streaming

and/or efficient access to all fragments when responding to subsequent requests.

§VI.A.4 (motivation); Ex-1003, ¶146.⁵

b. 2[b]/12[b]

Mattis-Hosur, with or without Sackstein, teaches/suggests 2[b]/12[b]. Ex-1003, ¶¶147-49.

Mattis discloses “the Directory Table is updated” when writing, moving, or deleting fragments (*segments*) in memory, (*transmitting...instructions to update...in response to storing the segment of the stitched optimized version*). Mattis, 18:65-19:5, 21:38-53 (writing), 23:37-52 (moving/deleting), 24:4-15. Hosur similarly updates locations in auxiliary for newly transcoded video portions. Hosur, ¶¶5-6, 46, 48-49, 53; Ex-1003, ¶148.

When implementing Mattis to transcode and store fragments/portions of a video on different transcoders consistent with Hosur and/or Sackstein (§VI.A.5.h (1[e]/11[g])), Mattis’s Tag/Directory Tables (*database*) would be updated when fragments/portions are moved to a single transcoder/transcoded-content store. §VI.A.6.a (2[a]/12[a]); §VI.A.4 (motivation); Ex-1003, ¶149.

⁵ If “a segment” may be stored *prior* to stitching, this claim is also met when transcoding new portions for subsequent stitching. §VI.A.11-VI.A.12 (cls. 7-8).

7. Claim 3/13

Mattis-Hosur, with or without Sackstein, teaches/suggests 3/13. Ex-1003, ¶¶150-52.

Mattis discloses a matching key provides the location(s) of fragments of a specific version of an object, and Mattis and Hosur (with or without Sackstein), teach storing different portions of a transcoded video in different transcoded-content stores. §§VI.A.5.g-VI.A.5.h (1[d]/11[f]-1[e]/11[g]). When implementing Mattis to transcode and store fragments/portions of a video on different transcoders consistent with Hosur and/or Sackstein (§VI.A.5.h (1[e]/11[g])), a response to a matching key in Mattis's Tag/Directory Tables would indicate that different fragments/portions (*segments*) of a transcoded video (*optimized version of the same original source file*) are stored in different transcoded-content stores (*a plurality of different video optimizers*). §VI.A.4 (motivation); Ex-1003, ¶¶151-52.

8. Claims 4/14

Mattis-Hosur, with or without Sackstein, teaches/suggests 4/14. Ex-1003, ¶¶153-55.

Although Mattis does not disclose how fragments distributed in different memory are retrieved for streaming, Hosur teaches that a streamer is notified that a transcoder has portion(s) of a requested video, which the streamer accesses and combines with other portions. Hosur, ¶¶49, 53-55. Because Mattis-Hosur, with or

without Sackstein, would store different portions of a video in different transcoded-content stores (§§VI.A.5.h (1[e]/11[g])), VI.A.4 (motivation)), and network systems operate on request-response principles, a network element (e.g., Mattis’s cache or Hosur’s streamer) would *request* fragments/segments from different transcoded-content stores to then chain/combine/splice them for streaming. Ex-1003, ¶¶154-55. Indeed, Sackstein expressly teaches this. Sackstein, ¶47 (“the controller 260 may also issue requests for the segment to the caches of other or neighboring systems”).

9. Claims 5/15

Mattis discloses/suggests claims 5/15. Ex-1003, ¶¶156-58.

Mattis discloses that object keys are a hash of a file’s contents, and name keys are a hash of a file’s name—each being a *source content signature being a hash of the original source file* (either the file itself, or its name). Mattis, 8:27-50, 8:64-9:15; *see* Ex-1001, 6:64-7:11; Ex-1003, ¶157. Alternatively, it would have also been obvious to use a hash of the contents of the original version of a file as an alternative to Mattis’s name key. Ex-1003, ¶158 (citing Ex-1029, ¶67).

10. Claims 6/16

Mattis-Hosur, with or without Sackstein, teaches/suggests claims 6/16. Ex-1003, ¶¶159-62.

Mattis discloses the Tag/Directory Tables (*database*) comprise reference-key entries (*data entries*) associated with the locations of different versions of a

requested object (or fragments). §§VI.A.5.f-VI.A.5.h (1[c]/11[e]-1[e]/11[g]). Mattis also discloses updating Tag/Directory Table entries when the location of an object (or fragments) changes. §VI.A.6.b (2[b]/12[b]); Ex-1003, ¶160. Hosur also teaches storing multiple versions of a video (Hosur, ¶60), and transcoding videos using different transcoders, which would thus be stored in different transcoded-content stores, and updating auxiliary files to provide their locations (*id.*; *see also id.*, ¶¶5-6, 46, 48, 53, 59, 78). Ex-1003, ¶161.

When combining Mattis and Hosur (with or without Sackstein) to store different versions of videos in different transcoded-content stores (§VI.A.5.h (1[e]/11[g])), the locations in Mattis's Tag/Directory Tables would be those of the transcoded-content stores storing those different versions, consistent with Hosur (*a plurality of video optimizers storing an optimized version of the original source file*). Hosur, Abstract, ¶¶5, 46, 53-54, 70-71, cl. 13, 28; §VI.A.4 (motivation); Ex-1003, ¶162.

11. Claims 7/17

Mattis-Hosur, with or without Sackstein, teaches/suggests claims 7/17. Ex-1003, ¶¶163-66.

Although Mattis does not disclose transcoding portions of video, Hosur and Sackstein both teach checking “whether content from the requested source media file 30 is already stored” (Hosur, ¶40; *see also* Sackstein, ¶58), and transcoding if

not (Hosur, ¶41; Sackstein, ¶58). Ex-1003, ¶164. Sackstein also expressly describes checking for, and otherwise transcoding, individual segments of video, and not just entire videos. Sackstein, ¶58; Ex-1003, ¶164.

Because the Mattis-Hosur combination, with or without Sackstein, provides multiple video transcoders capable of transcoding individual video portions/segments upon request (§§VI.A.5.d (1[a]/11[c]), VI.A.5.h (1[e]/11[g])), some portions of an already-transcoded video would be stored in different transcoded-content stores, but others may be missing, and any missing portions would be transcoded and chained/combined/spliced to generate a complete transcoded video. §VI.A.4 (motivation); Ex-1003, ¶165 (citing Ex-1030, ¶¶42, 63, Figs. 11a-b). Hosur also discloses a single transcoder may transcode multiple portions of a video (Hosur, ¶¶44, 46-48), and thus in some instances newly transcoded portions would be stored in the same transcoded-content store as a first portion (*storing the transcoded second segment in the local cache of the video optimizer*) (Ex-1003, ¶166).

12. Claims 8/18

Mattis-Hosur, with or without Sackstein, teaches/suggests claims 8/18. §§VI.A.6.b (2[b]/12[b]), VI.A.11 (cl.7); Ex-1003, ¶167.

13. Claims 9/19

Mattis-Hosur, with or without Sackstein, teaches/suggests claims 9/19. Ex-1003, ¶¶168-72.

Mattis-Hosur, with or without Sackstein, provides multiple video transcoders capable of transcoding individual video portions upon request. (§§VI.A.5.d (1[a]/11[c]), VI.A.5.h (1[e]/11[g])). And both Hosur and Sackstein check whether transcoded portions/segments are already stored and transcoding missing portions/segments. §VI.A.11 (cl.7); Ex-1003, ¶169. Accordingly, when a portion of untranscoded video (*second segment*) falls between two transcoded portions (*first portion and third segment*), the untranscoded portion would be transcoded from the start of the second segment to the start of the third segment (*transcoding the original source file from a start of the second segment to a start of the third segment*) to generate a complete transcoded video. §§VI.A.5.h (1[e]/11[g]), VI.A.11 (cl.7); VI.A.4 (motivation); Ex-1003, ¶¶170-71 (citing Ex-1030, ¶¶42, 63, Figs. 11a-b).

It would have further been understood based on Hosur and/or Sackstein that (some) already-transcoded segments would be stored in different transcoded-content stores (*third segment...that is stored by one of the plurality of other video optimizers*). §§VI.A.5.h (1[e]/11[g]), VI.A.11 (cl.7); VI.A.4 (motivation); Ex-1003, ¶172.

14. Claims 10/20

Mattis-Hosur, with or without Sackstein, teaches/suggests claims 10/20. Ex-1003, ¶¶173-76.

Mattis discloses a “garbage collection” process that deletes fragments (*purges*) based on frequency of access. Mattis, 21:54-23:52. A POSITA would have expected that first portions/fragments of a video would be accessed more frequently than later fragments (e.g., starting but not completing playback of an entire video), and thus in many instances later fragments would be deleted from the transcoded-content store but the first portion would not. §VI.A.4 (motivation); Ex-1003, ¶¶175-76 (citing Ex-1031, ¶34). This is also consistent with Sackstein, which teaches deleting individual segments for memory management, but not all segments. Sackstein, ¶¶45, 59; Ex-1003, ¶175.

B. Ground 2A: Hosur in view of Mattis, with or without Sackstein

1. Motivation to Combine

A POSITA would have also been motivated to modify Hosur in view of Mattis and Sackstein. §VI.A.4 (Ground 1, same field); Ex-1003, ¶177.

Hosur stores and locates different version of video (or portions) using “auxiliary files.” Hosur, ¶¶5-6, 46-48, 53-56, 61-64. Each auxiliary file is stored with its associated video file, and thus auxiliary files are distributed throughout the network. *Id.*, ¶¶34-35. Accordingly, Hosur searches different content stores for

relevant auxiliary files, and then searches the auxiliary files to determine whether they are responsive to a request for a specific version of video. Ex-1003, ¶178.

If Hosur's sets of auxiliary files do not constitute a "database," as claimed (§VI.B.2.f (1[c]/11[e])), rather than use distributed auxiliary files to locate relevant video portions, Mattis uses object/name keys and a centralized database of Tag/Directory Tables. *See* §§VI.A.1 (Mattis Overview), VI.A.5.e-VI.A.5.f (1[b]/11[d]-1[c]/11[e]); Ex-1003, ¶179.

A POSITA would have been motivated to supplement or replace Hosur's distributed auxiliary files with a centralized database of keys like Mattis's for several reasons. Ex-1003, ¶¶180, 183. First, Hosur does not provide a mechanism to collect all versions of a requested video, or de-duplicate identical videos. Ex-1003, ¶180. Mattis explains that name and object keys provide these features, respectively, thus reducing memory and processing requirements of the system. Mattis, 8:51-9:15, 10:10-22; Ex-1003, ¶181.

Second, querying a centralized database like Mattis's Tag/Directory Tables would be simpler than searching different content stores for auxiliary files. Ex-1003, ¶181.

Finally, this would have been a simple substitution of one known element (distributed auxiliary files) for another (centralized database), or replacing one known technique (distributed searching) with another suitable option (centralized

searching) to achieve a predictable result or solve a known problem in the art (locating a specific version of a requested video). *Id.*, ¶182; *Intel Corp. v. PACT XPP Schweiz AG*, 61 F.4th 1373, 1380 (Fed. Cir. 2023); *Snap Inc. v. You Map, Inc.*, No. 2024-1120, 2025 WL 2793538, *3-4 (Fed. Cir. Oct. 1, 2025).

A POSITA would have had a reasonable expectation of success in making this modification because Mattis's keys and tables represent similar content (file parameters and locations) and serve the exact same purpose (to locate specific versions of a file) as Hosur's auxiliary files. Ex-1003, ¶184. Indeed, searching Hosur's auxiliary files is materially the same as searching by Mattis's name keys, except for where the reference parameters are stored (auxiliary files versus a database). *Id.* The only modifications required would be to build centralized tables for searching, rather than implement distributed search algorithms as Hosur's disclosure implies. *Id.*

Moreover, a POSITA would have also been motivated to combine Hosur, Mattis, and Sackstein to enable concurrent transcoding and distributed storage for the same reasons discussed with respect to Mattis, Hosur, and Sackstein. §VI.A.4 (Ground 1, motivation); Ex-1003, ¶¶185-86.

2. Claims 1/11

a. 1[pre]/11[pre]

If limiting, Hosur discloses or suggests 1[pre]/11[pre]. Ex-1003, ¶¶187-88.

Hosur discloses a “media-content delivery system” (Hosur, ¶¶1, 14, Figs. 1, 3-4), and “method for delivering media content...to a remote client” (Hosur, Abstract; *see id.*, ¶¶6-7, 14, Fig. 1 (showing client devices 12)). Hosur explains that “clients 12 may be capable of receiving and/or playing [different versions of] media,” (*id.*, ¶18), and Hosur provides a way to transcode videos “for transmission to and display by the requesting client 12” (*streaming optimized video*), (*id.*, ¶¶15, 21, 25-26). Ex-1003, ¶188. Hosur, ¶¶73-75 (transcoders including processor(s) and memory), 81-82 (other network elements may be hardware and/or “software applications being executed by [a] processor”), 83-85 (streamers including processor(s) and memory).

b. 11[a]

Hosur discloses/suggests 11[a]. Ex-1003, ¶¶189-90.

Hosur discloses the system comprises *at least one processor*. Hosur, ¶¶73-75, 81-85, 89, cl. 13-23, 31-37, Figs 3-4; Ex-1003, ¶190.

c. 11[b]

Hosur discloses/suggests 11[b]. Ex-1003, ¶¶191-92.

Hosur discloses the system comprises memory (*non-transitory computer-readable storage medium*) for storing *executable computer program instructions to be executed by the processor*. Hosur, ¶¶31, 34, 43, 73, 75, 80, 82-83, 85, 90, Figs. 3-4. The instructions would cause the processor to perform each of the functions

addressed below, either alone or in combination with Mattis and/or Sackstein. Ex-1003, ¶¶192.

d. 1[a]/11[c]

Hosur, alone or combined with Mattis, teaches/suggests 1[a]/11[c]. *Id.*, ¶¶193-96.

Hosur discloses that “clients 12 may include...computers.” Hosur, ¶17; *id.*, ¶18. Hosur also discloses “receiving a request identifying a requested media file” (*receiving a request [for an] original source file from a client computing device*). Hosur, ¶¶5, 21 (“CMS 16 processes requests from clients 12”); Ex-1003, ¶194.

“Transcoders 20 may be capable of modifying content in source media files 30...from a first format...to a second format, such as a target format appropriate for transmission to and display by a requesting client.” Hosur, ¶26. “Content request 42 may identify a specific source media file 30” (*id.*, ¶38), and “indicate the appropriate format that the requested content should have,” such as “specifying the codec, bitrate, and/or other parameters” (*the request including a key specifying parameters to optimize the original source file*) (*id.*, ¶39). “In response..., CMS 16 may...determine an appropriate format (referred to here as the ‘target format’) for the requested content to allow, facilitate, and/or optimize transmission to and playback by the requesting client” (*a request to optimize*). *Id.*, ¶39; Ex-1003, ¶195.

Mattis includes object/name keys in requests to locate a specific versions of multimedia objects (§VI.A.5.d (Ground 1, 1[a]/11[c])), just like Hosur. When supplementing/replacing Hosur's auxiliary files with object/name keys and a database like Mattis's Tag/Directory Tables, the object/name keys in a request would be used to locate a specific version of a video. §VI.B.1 (motivation); Ex-1003, ¶196.

e. 1[b]/11[d]

Hosur, alone or combined with Mattis, teaches/suggests 1[b]/11[d]. Ex-1003, ¶¶197-03.

Hosur discloses two ways to locate a first portion of a video.

First, CMS and/or transcoder determines whether a specific version of a video (or portion) is already stored. Hosur, ¶¶40, 95. Hosur searches cached-content stores as one example (*id.*), but Hosur would have been understood to also search transcoded-content stores because: (1) transcoded-content stores store videos too and the system knows what videos they store (*id.*, ¶33, 58-59, 95); and (2) although cached-content stores are intended for longer-term storage (*id.*, ¶¶35-36), they are optional (*id.*, ¶41). Ex-1003, ¶199. Hosur therefore would have been understood to search transcoded-content stores (*a local cache of a video optimizer*) for a first portion of transcoded video (*identifying... a first portion of the optimized version of the original source file*). Ex-1003, ¶200; *see also* Mills, Fig. 3A, ¶61 (expressly searching both cached-content stores and transcoded-content stores). As explained

below, Hosur uses the request's parameters to search for a specific version of video (or portions) (*based on the key*). §§VI.B.2.f-VI.B.2.h (1[c]/11[e]-1[e]/11[g]); Ex-1003, ¶201.

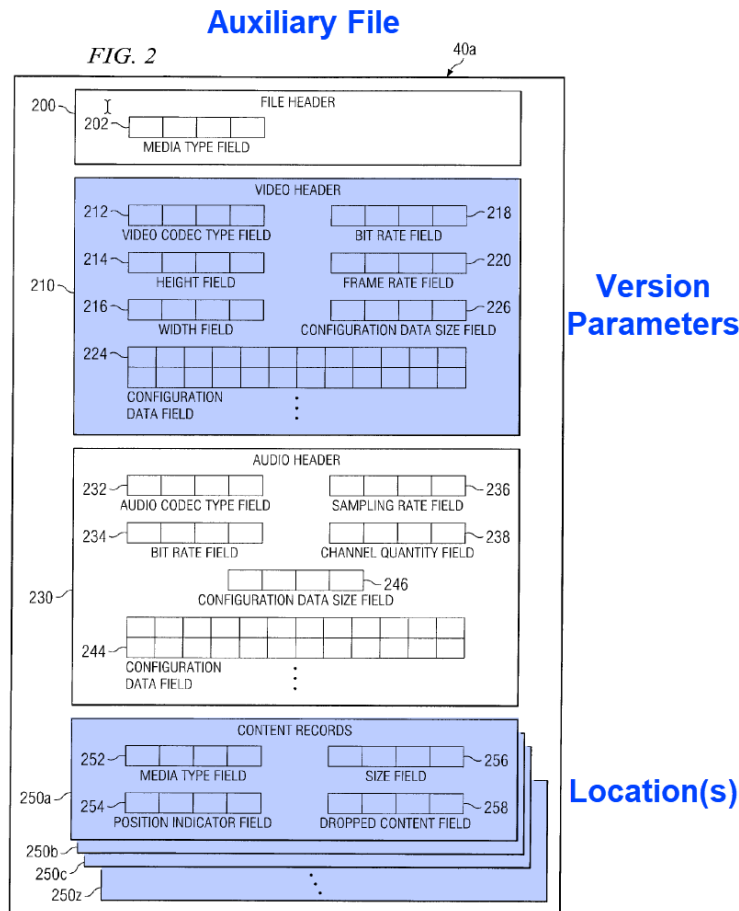
Second, if the stores do “not contain a cached media file 34...in the target format..., CMS 16 may initiate transcoding” into the target format (Hosur, ¶¶41) based on the parameters (*based on the key*). *Id.*, ¶39 (“content request 42 may indicate the appropriate format”). Transcoders store transcoded portions in a local transcoded-content store (*local cache of a video optimizer*). *Id.*, ¶¶34, 44. “[T]ranscoder 20 may generate a partial auxiliary file 40a...containing [location/parameter] information about a first portion of transcoded media file 32a,” which streamers may “incrementally” access to stream that portion (*identifying...a first portion*). *Id.*, ¶48; Ex-1003, ¶202.

Mattis also teaches using object/name keys in requests to locate specific versions of a multimedia object (or “fragments”) (§VI.A.5.e (Ground 1, 1[b]/11[d])), just like Hosur's requests and auxiliary files, (Ex-1003, ¶203). When supplementing/replacing Hosur's auxiliary files with object/name keys and a database like Mattis's Tag/Directory Tables, object/name keys would be used to locate specific versions of video (including portions) in Hosur's transcoded-content stores, consistent with Mattis. §VI.B.1 (motivation); Ex-1003, ¶203.

f. 1[c]/11[e]

Hosur, alone or combined with Mattis, teaches/suggests 1[c]/11[e]. Ex-1003, ¶¶204-09.

After receiving a request, Hosur queries auxiliary files to determine whether the requested version of video (or portions) is already stored, or has since been generated by a transcoder (*generating a query*) using the request's parameters (*based on the key*). §VI.B.2.e (1[b]/11[d]); Ex-1003, ¶205. Each auxiliary file identifies: (1) the locations of different portions a specific version (Hosur, ¶¶5-6, 46); and (2) the parameters associated with that specific version (*id.*, ¶¶46, 62-64) (*reference keys...specifying parameters of an optimized version of a source*). Ex-1003, ¶206.



Hosur, Fig. 2

“[T]ranscoded...content store...store[s] auxiliary files 40 associated with the relevant transcoded media files” (Hosur, ¶¶34, 35), constituting a *database* of media files/auxiliary files (*a plurality of reference keys corresponding to a plurality of optimized versions of source files*). Ex-1003, ¶207. Moreover, because auxiliary files “indicate a location of the associated portion of media content” (Hosur, ¶46), and there can be multiple transcoded-content stores (Hosur, ¶31), auxiliary files may identify other transcoded-content stores where portions of a video are being stored

(reference keys corresponding to a plurality of optimized versions of source files stored on a plurality of other video optimizers). Ex-1003, ¶208.

Mattis also teaches using the object/name key in a request to query the Tag/Directory Tables to locate a specific version of a multimedia object (or “fragment”), (§VI.A.5.f (Ground 1, 1[c]/11[e])), just like Hosur’s parameters and auxiliary files (Ex-1003, ¶209). When supplementing/replacing Hosur’s auxiliary files with object/name keys and a database like Mattis’s Tag/Directory Tables, the database would be queried for keys matching a request’s object/name keys to locate a specific version of video (or portions/fragments), consistent with Mattis. §VI.B.1 (motivation); Ex-1003, ¶209.

g. 1[d]/11[f]

Hosur, alone or combined with Mattis, teaches/suggests 1[d]/11[f]. Ex-1003, ¶¶210-13.

Hosur discloses that both requests and auxiliary files identify specific versions of video by their parameters (e.g., codec, bitrate, resolution) (Hosur, ¶39, 46), that CMS or transcoders use to “determine whether content...is already stored...in the target format” (*id.*, ¶¶40, 95; §VI.B.2.e (1[b]/11[d])). A POSITA would have understood that CMS or transcoder *receives* an indication (*response*) from the transcoded-content store (*database*) whether an auxiliary file has parameters matching those in the request (*a response indicating whether the key matches a*

reference key) to “determine whether content...is already stored...in the target format.” Hosur, ¶¶40, 95; Ex-1003, ¶¶211-12.

Mattis also teaches receiving a response from the Tag/Directory Tables indicating whether the Tables have a matching key (§VI.A.5.g (Ground 1, 1[d]/11[f])), just like Hosur’s parameters and auxiliary files. When supplementing/replacing Hosur’s auxiliary files with object/name keys and databases like Mattis’s Tag/Directory Tables, Hosur would receive responses (e.g., at CMS, transcoder, or streamer) indicating whether a request’s key matches one in the database. §VI.B.1 (motivation); Ex-1003, ¶213.

h. 1[e]/11[g]

Hosur, with or without Mattis and Sackstein, teaches/suggests 1[e]/11[g]. Ex-1003, ¶¶214-19.

Hosur suggests that, during an initial search, request and auxiliary-file parameters are compared to locate existing portions of a video already stored in content stores. §VI.B.2.g (1[d]/11[f]); Ex-1003, ¶215. If transcoding is required, however, Hosur discloses that “[s]treamer 28 receives...a first record in [the] auxiliary file” that provides “location information to access...first media portion,” and “may repeat this process for additional records” to “transmit [additional] portions” (*receiving from the database...the address...storing a segment of the*

optimized version of the original source file). Hosur, ¶¶53-54; *id.*, 70-71; Ex-1003, ¶216.

Hosur also discloses storing and serving different videos (or portions) from different transcoded-content stores. Hosur, ¶¶31 (“content stores may each represent a plurality of devices and may be distributed across multiple locations”), 33 (storing “partial file[s]”), 46 (locating portions), 48-49, 53, 55, 78. But Hosur also envisions transcoding and storing different portions of a single video in different transcoded-content stores. §VI.B.1 (motivation) (citing by reference Hosur, ¶¶15, 31, 34, 78). Accordingly, when different portions of a video are stored in different transcoded-content stores, the auxiliary file with matching parameters would provide the locations of the different transcoded-content stores (*receiving...in response to a reference key matching the key, the address of another video optimizer*). §VI.B.1 (motivation); Ex-1003, ¶217.

Regardless, Sackstein teaches transcoding and storing different portions of a video in different transcoders. §VI.B.1 (motivation) (citing by reference Sackstein, ¶¶22, 28-35, 39-40, 47, 56-58). When expressly enabling Hosur to transcode and store different portions of a video in different transcoders, consistent with Sackstein, the *address* received from the auxiliary files would be the address of *another video optimizer storing a segment of the optimized version of the original source file*. *Id.*; Ex-1003, ¶218.

Mattis also teaches that, when a request's key matches a key in the Tag/Directory Tables, a location of the multimedia object (or "fragment") is provided. §VI.A.5.h (Ground 1, 1[e]/11[g]). When supplementing/replacing Hosur's auxiliary files with object/name keys and databases like Mattis's Tag/Directory Tables, the object/name key in the database would identify the address of the different transcoded-content stores (*video optimizers*) storing different fragments/portions (*storing a segment*) of the requested version of video of Hosur and/or Sackstein. §VI.B.1 (motivation); Ex-1003, ¶219.

i. 1[f]/11[h]

Hosur, with or without Sackstein, teaches/suggests 1[f]/11[h]. Ex-1003, ¶¶220-23.

Hosur suggests storing different transcoded portions of a video in different transcoded-content stores. §VI.B.1 (motivation) (citing by reference Hosur, ¶¶15, 31, 34, 78). Hosur also discloses combining portions into a dynamic "transcoded media file" that "allow[s] streaming of content...while transcoded media file 32a...[is] still being written," (Hosur, ¶48), and "streamers 28...synchronize portions...before streaming the relevant portions to client 12" (*id.*, ¶55). *See also id.*, ¶¶46-47, 53-56; Ex-1003, ¶221. Hosur thus contemplates "synchronizing" different portions of a video from different transcoded-content stores into a "transcoded media file"

(stitching the first portion...with the segment of the optimized version of the original source file stored on the other video optimizer). Ex-1003, ¶221.

Regardless, Sackstein expressly teaches storing different video portions in different transcoders, searching different transcoder caches for portions of a video, and “splicing” them together for streaming to the client. §VI.B.1 (motivation) (citing by reference Sackstein, ¶¶22, 28-35, 39-40, 47, 56-58). When transcoding and storing different portions of a video on different transcoders consistent with Hosur and/or Sackstein (§VI.B.2.h (1[e]/11[g])), the portions would be collected from the different transcoders and combined/spliced together for streaming. §VI.B.1 (motivation); Ex-1003, ¶¶222-23.

j. 1[g]/11[i]

Hosur, with or without Sackstein, teaches/suggests 1[g]/11[i]. Ex-1003, ¶¶224-25.

Hosur “synchronize[s] portions...before streaming the relevant portions to client 12...for playback” (*streaming the stitched optimized version...for playback on the client device*). Hosur, ¶55. Sackstein teaches the same. Sackstein, ¶¶33 (playback), 40-41 (splicing for streaming); §VI.C.2 (motivation); Ex-1003, ¶225.

3. Claims 2/12

a. 2[a]/12[a]

Hosur, with or without Sackstein, teaches/suggests 2[a]/12[a]. Ex-1003, ¶¶226-28.

Hosur, with or without Sackstein, discloses combining different portions of transcoded video from different transcoders into a “transcoded media file.” §VI.B.2.i (1[f]/11[h]); Ex-1003, ¶227. A POSITA would have understood that, to form the combined file, the different portions would either be copied to a single transcoded-content store *before* streaming, or the combined “transcoded media file” generated by, e.g., a streamer would be copied to a transcoded-content store *after* streaming so all portions would be stored together (*storing a segment...in the cache of the video optimizer*) for responding to subsequent requests for the same transcoded video. Ex-1003, ¶227 (citing Ex-1033, ¶85). Indeed, Hosur envisions “streamer 28 may increment a pointer” to stream different portions within the file, meaning different portions may ultimately be moved into sequential pieces of memory. Hosur, ¶54; Ex-1003, ¶228.⁶

⁶ If “a segment” may be stored *prior* to stitching, this claim is also met when transcoding new portions for subsequent stitching. §§VI.B.8-VI.B.9 (cls. 7-8).

b. 2[b]/12[b]

Hosur, with or without Mattis and Sackstein, teaches/suggests 2[b]/12[b]. Ex-1003, ¶¶229-32.

Hosur updates an auxiliary file as portions are transcoded (Hosur, ¶48), and also enables other transcoders to write auxiliary files on behalf of another transcoder (*id.*, ¶47). Moreover, Hosur suggests that different transcoders may transcode and store different portions of a transcoded video in different transcoded-content stores. §VI.B.1 (motivation); Ex-1003, ¶230.

To enable these features, transcoders must communicate information (like the location of each portion) that needs to be written into the auxiliary file (*transmitting...instructions to update...in response to storing the segment of the stitched optimized version of the original source file stored in the video optimizer*), including updating existing auxiliary files in the transcoded-content stores (*database*). Ex-1003, ¶231.

Mattis also teaches updating Tag/Directory Tables when writing, moving, or deleting fragments of files. §VI.A.6.b (Ground 1, 2[b]/12[b]). When supplementing/replacing Hosur's auxiliary files with object/name keys and databases like Mattis's Tag/Directory Tables, the database would be updated when collecting all portions together in a "transcoded media file." §§VI.B.3.a (2[a]/12[a]), VI.B.1 (motivation); Ex-1003, ¶232.

4. Claim 3/13

Hosur, with or without Mattis and Sackstein, teaches/suggests claim 3/13. Ex-1003, ¶¶233-35.

Hosur, with or without Sackstein, teaches that different portions of a video may be stored in different transcoded-content stores. §VI.B.1 (motivation). Accordingly, when querying auxiliary files for a video, the content records would identify different transcoders/transcoded-content stores storing the different portions of video. §VI.B.2.h (1[e]/11[g]); Ex-1003, ¶234.

Mattis also teaches using a matching object key in Tag/Directory Tables to locate all fragments of a requested object. §VI.B.2.h (1[e]/11[g]). When supplementing/replacing Hosur's auxiliary files with object/name keys and databases like Mattis's Tag/Directory Tables, a response to a matching key in the database would indicate that different fragments/portions (*segments*) of a transcoded video (*optimized version of the same original source file*) are stored in different transcoded-content stores (*a plurality of different video optimizers*). §VI.B.1 (motivation); Ex-1003, ¶235.

5. Claims 4/14

Hosur, with or without Sackstein, teaches/suggests claim 4/14. Ex-1003, ¶¶236-38.

Hosur discloses that a streamer is notified when a transcoder has portion(s) of a requested video, which streamer accesses and combines with other portions for streaming. Hosur, ¶¶49, 53-55; §VI.B.2.i (1[f]/11[h]). Because Hosur, with or without Sackstein, stores different portions of a video in different transcoded-content stores (§VI.B.1 (motivation)), and network systems operate on request-response principles, Hosur's streamer would be understood to *request* portions from different transcoded-content stores to then collect and combine into the "transcoded media file" for streaming. Ex-1003, ¶¶237-38. Indeed, Sackstein expressly teaches this. Sackstein, ¶47 ("the controller 260 may also issue requests for the segment to the caches of other or neighboring systems").

6. Claims 5/15

Hosur-Mattis teaches/suggests claims 5/15. Ex-1003, ¶¶239-41.

Although Hosur does not expressly disclose hashed keys, Mattis teaches that object keys are a hash of a file's contents, and name keys are a hash of a file's name—each being a *source content signature being a hash of the original source file* (either the file itself, or its name). Mattis, 8:27-50, 8:64-9:15; Ex-1001, 6:64-7:11); Ex-1003, ¶240. It would have been obvious to implement Mattis's keys in Hosur's systems. §VI.B.1 (motivation); Ex-1003, ¶240. Alternatively, it would have also been obvious to use a hash of the contents of the original version of a file as an alternative to Mattis's name key. Ex-1003, ¶241 (citing Ex-1029, ¶67).

7. Claim 6/16

Hosur, alone or combined with Mattis, teaches/suggests claims 6/16. Ex-1003, ¶242-244.

Hosur discloses storing multiple versions of a video (Hosur, ¶60), and transcoding videos using different transcoders, which would thus be stored in different transcoded-content stores (*a plurality of optimization server storing an optimized version of the original source file*) (*id.*, ¶¶5-6, 46, 48, 53, 59, 78). Hosur also discloses that auxiliary files are stored in transcoded-content stores (*database*). §VI.B.2.f (1[c]); Ex-1003, ¶243. The “auxiliary file includes a plurality of content records [*(data entries)*] associated with different portions of the media content in the modified media file and indicating a location of the associated portions” (*identifying*). Hosur, ¶5; *id.*, Abstract, ¶¶46, 53-54, 70-71, cl. 13, 28. Finally, Hosur updates content records to identify the location of new transcoded portions as they are generated (*entries being modified....*). §VI.B.3.b (2[b]); Ex-1003, ¶243.

Mattis also teaches the Tag/Directory Tables (*database*) comprise reference-key entries (*data entries*) associated with the locations of different versions of a requested object (or fragments), and updating Tag/Directory Table entries when the location of objects (or fragments) changes. §VI.A.10 (Ground 1, cl. 6). When supplementing/replacing Hosur’s auxiliary files with object/name keys and databases like Mattis’s Tag/Directory Tables, the locations in the database would be

those of the transcoded-content stores storing different versions of a video, consistent with Hosur (*a plurality of optimization servers storing an optimized version of the original source file*). §VI.B.1 (motivation); Ex-1003, ¶244.

8. Claims 7/17

Hosur, with or without Sackstein, teaches/suggests claims 7/17. Ex-1003, ¶¶245-48.

Hosur and Sackstein both teach checking “whether content from the requested source media file 30 is already stored” (Hosur, ¶40; *see also* Sackstein, ¶58), and transcoding if not (Hosur, ¶41; Sackstein, ¶58). Ex-1003, ¶246. Sackstein also expressly describes checking for, and otherwise transcoding, individual segments of video, and not just entire videos. Sackstein, ¶58; Ex-1003, ¶246.

Because Hosur, with or without Sackstein, provides multiple transcoders capable of transcoding individual video portions upon request (§VI.B.2.h (1[e]/11[g])), some portions of an already-transcoded video would be stored in different transcoded-content stores, but others may be missing, and any missing portions would be transcoded and combined with existing portions to generate a complete transcoded video, as was known. §VI.B.1 (motivation); Ex-1003, ¶247 (citing Ex-1030, ¶¶42, 63, Figs. 11a-b). Hosur also discloses a single transcoder may transcode multiple portions of a video (Hosur, ¶¶44, 46-48), and thus in some instances newly transcoded portions would be stored in the same transcoded-content

store as a first portion (*storing the transcoded second segment in the local cache of the video optimizer*) (Ex-1003, ¶248).

9. Claims 8/18

Hosur-Mattis, with or without Sackstein, teaches/suggests claims 8/18. §§VI.B.3.b (2[b]/12[b]), VI.B.8 (cl. 7); Ex-1003, ¶249.

10. Claims 9/19

Hosur, with or without Sackstein, teaches/suggests claims 9/19. §VI.B.8 (cl.7); Ex-1003, ¶¶250-54.

Hosur, with or without Sackstein, provides multiple video transcoders capable of transcoding individual video portions upon request. (§VI.B.2.h (1[e]/11[g])). And both Hosur and Sackstein check whether transcoded portions/segments are already stored, and transcoding missing portions/segments. §VI.B.8 (cl.7); Ex-1003, ¶251. Accordingly, when a portion of untranscoded video (*second segment*) falls between two transcoded portions (*first portion and third segment*), the untranscoded portion would be transcoded from the start of the second segment to the start of the third segment (*transcoding the original source file from a start of the second segment to a start of the third segment*) to generate a complete transcoded video, as was known. §§VI.B.2.h (1[e]/11[g]), VI.B.8 (cl.7); VI.B.1 (motivation); Ex-1003, ¶¶252-53 (citing Ex-1030, ¶¶42, 63, Figs. 11a-b).

It would have further been understood based on Hosur and/or Sackstein that (some) already-transcoded segments would be stored in different transcoded-content stores (*third segment...that is stored by one of the plurality of other video optimizers*). §§VI.B.2.h (1[e]/11[g]), VI.B.8 (cl.7); VI.B.1 (motivation); Ex-1003, ¶254.

11. Claims 10/20

Hosur-Mattis, with or without Sackstein, teaches/suggests claims 10/20. Ex-1003, ¶¶255-59.

Although Hosur discloses deleting video, Hosur does not describe memory management for stored videos. Hosur, ¶59; Ex-1003, ¶256.

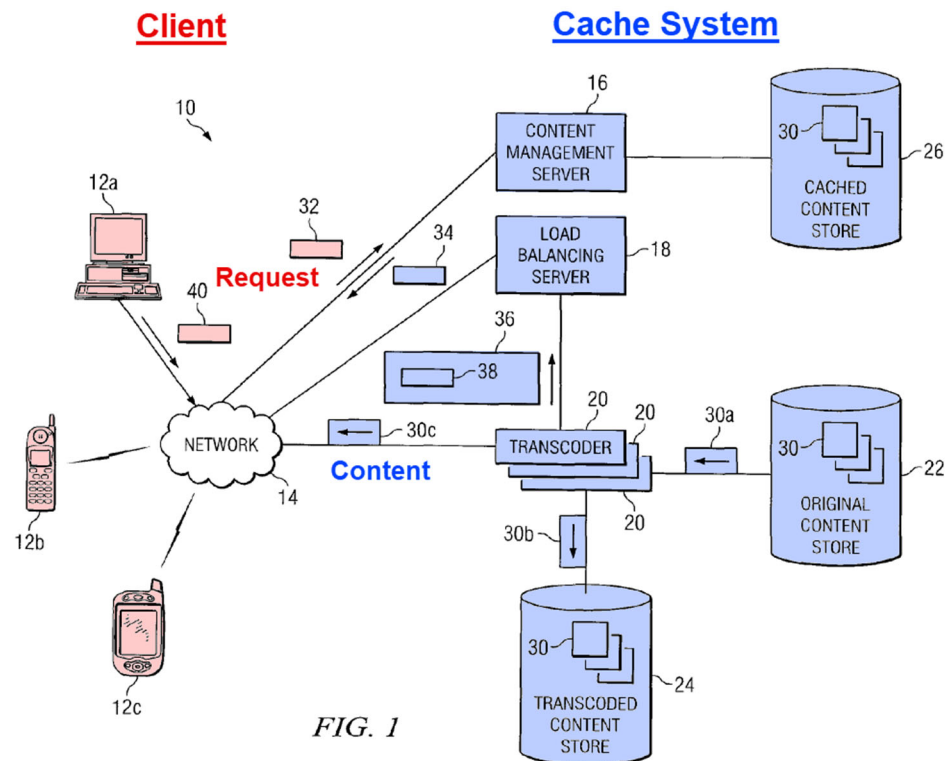
Mattis teaches a “garbage collection” process that deletes fragments (*purges*) based on frequency of access. Mattis, 21:54-23:52. A POSITA would have been motivated to modify Hosur to implement a “garbage collection” process like Mattis’s to manage available memory in Hosur’s transcoders (*video optimizers*). Ex-1003, ¶¶257-58. In doing so, a POSITA would have expected that first portions/fragments of a video would be accessed more frequently than later fragments (e.g., starting but not completing playback of an entire video), and thus in many instances later fragments would be deleted from the transcoded-content store but the first portion would not. §VI.B.1 (motivation); Ex-1003, ¶¶258-59 (citing Ex-1031, ¶34). This is also consistent with Sackstein, which teaches deleting individual

segments for memory management, but not all segments. Sackstein, ¶¶45, 59; Ex-1003, ¶259.

C. Ground 3A: Mills in view of Mattis, with or without Sackstein

1. Mills

Mills discloses systems and methods “for managing media files having different format characteristics,” using nearly the same structures as Hosur. Mills, Abstract, ¶¶1, 12; Ex-1003, ¶¶68-70.



Mills, Fig. 1

Like Hosur, Mills (1) stores and delivers different versions of videos (or portions) (Mills, ¶¶5, 13-15, 21, 23, 26); (2) generates new versions (or portions) upon request

using transcoders (*id.*, ¶¶28-31, 37-40); and (3) streams transcoded portions as they become available (*id.*, ¶¶41-44). *See also id.*, Figs. 3A-B; Ex-1003, ¶69.

Mills differs from Hosur in two predominant ways. First, although Mills (like Hosur) includes video parameters in requests (Mills, ¶33), Mills does not detail how those parameters are used to locate a specific version of a video (like Hosur's auxiliary file). Ex-1003, ¶70. Second, although Hosur would have been understood this way (§VI.B.2.f (Ground 2, 1[c])), Mills expressly discloses searching for portions of video stored in different transcoded-content stores. Mills, ¶61, Fig. 3A (at 308); Ex-1003, ¶70.

2. Motivation to Combine

Mills, Mattis, and Sackstein are all analogous art to the '442 patent and each other, because each is directed to networks for storing and delivering different versions of content to clients, and a POSITA would have been motivated to combine them. Ex-1003, ¶¶260-62.

Although Mills uses parameters in a request to identify a specific version of a video to send to a client, Mills does not detail exactly how the parameters are used to do so, leaving these implementation details to skilled artisans. *Id.*, ¶263.

Mattis would have been recognized as providing one obvious option to implement Mills's parameter-based identification. *Id.*, ¶264. Specifically, Mattis describes a system that—like Mills—also uses parameters in a request as part of a

“name key.” *See* §§VI.A.1 (Mattis Overview), VI.A.5.e-VI.A.5.f (Ground 1, 1[b]/11[d]-1[c]/11[e]); Ex-1003, ¶264. But, unlike Mills, Mattis describes how parameters are used to determine whether a file already exists in the requested format: by comparing name keys and parameters to reference name keys and parameters in Tag/Directory Tables. *See* §§VI.A.1 (Mattis Overview), VI.A.5.e-VI.A.5.f (Ground 1, 1[b]/11[d]-1[c]/11[e]); Ex-1003, ¶264. Moreover, Mattis provides an additional option—an “object key”—to detect duplicate objects and request a specific version of a specific file directly. *See* §§VI.A.1 (Mattis Overview), VI.A.5.e-VI.A.5.f (Ground 1, 1[b]/11[d]-1[c]/11[e]); Ex-1003, ¶264.

A POSITA would have been motivated to combine Mills and Mattis to implement Mattis’s object/name keys—and include object/name keys in client requests—as an obvious option to locate specific versions of videos in a way that is already consistent with Mills. Ex-1003, ¶265. Moreover, a POSITA would have recognized that Mattis’s keys and databases enable identifying *all* available versions of a requested object, as well as de-duplicating identical objects, thus reducing memory and processing requirements. Mattis, 8:51-9:15, 10:10-22; Ex-1003, ¶265.

Making this change would have been a simple substitution of one known element (Mills’s summary use of “parameters”) for another (Mattis’s detailed use of object/name keys and Tag/Directory Tables), or replacing one known technique (Mills’s use of parameters to identify specific versions of video) with another

suitable option (Mattis’s use of name/object keys and Tag/Directory Tables) to achieve a predictable result or solve a known problem in the art (locating a specific version of a requested file). Ex-1003, ¶266; *Intel Corp.*, 61 F.4th at 1380; *Snap Inc.*, 2025 WL 2793538, *3-4.

A POSITA would have had a reasonable expectation of success in making this modification because both references already suggest using what Mattis calls a “name key” to identify the requested video, as well as the parameters associated with a specific version of the requested video. Ex-1003, ¶267. All Mattis provides is additional details to carry out a search using those parameters, as well as an alternative type of (object) key to request specific versions directly. *Id.*, ¶267. The only modifications required would be to incorporate keys, databases, and queries based on Mattis’s teachings, which would have been well within the skills of a POSITA. *Id.*, ¶267.

Finally, like Hosur, Mills envisions multiple transcoders (Mills, ¶38)—each with transcoded-content stores (*id.*, ¶23)—operating “concurrently” (*id.*, ¶53); Ex-1003, ¶268. Mills also explains that transcoded-content stores may store “portions of media files 30” (Mills, ¶23) and transcoders may provide “faster than realtime transcoding of media files 30” (*id.*, ¶24). Ex-1003, ¶268. Thus, although a POSITA would have understood that Mills envisions storing different portions of a transcoded video in different transcoded-content stores (like Hosur) (Ex-1003, ¶¶269-71), a

POSITA would have been motivated to combine Mills, Mattis, and Sackstein to expressly do so as part of a concurrent transcoding scheme for the same reasons discussed with respect to Mattis, Hosur, and Sackstein. §VI.A.4 (Ground 1, motivation); Ex-1003, ¶¶272-73.

3. Claim 1/11

a. 1[pre]/11[pre]

If limiting, Mills discloses/suggests 1[pre]/11[pre]. Ex-1003, ¶¶274-75.

Mills discloses a “system and method for managing and modifying media content” (Mills, ¶1), and “delivering content from media files...to clients 12...of different reception and display requirements,” including video (*a method for streaming optimized video to a client device*) (Mills, ¶12; *id.*, ¶¶14-15, 21, 26, 43-45 (streaming), Figs. 3A-B). Ex-1003, ¶275. Mills also explains the method may be implemented by a computer. Mills, ¶¶49-50, 57 (transcoders including processor(s) and memory), 56 (other network elements may be hardware and/or “software applications being executed by [a] processor”); Ex-1003, ¶275.

b. 11[a]

Mills discloses/suggests 11[a]. Ex-1003, ¶¶276-77.

Mills discloses the system comprises *a processor*. Mills, ¶¶49-50, 56-57; Ex-1003, ¶277.

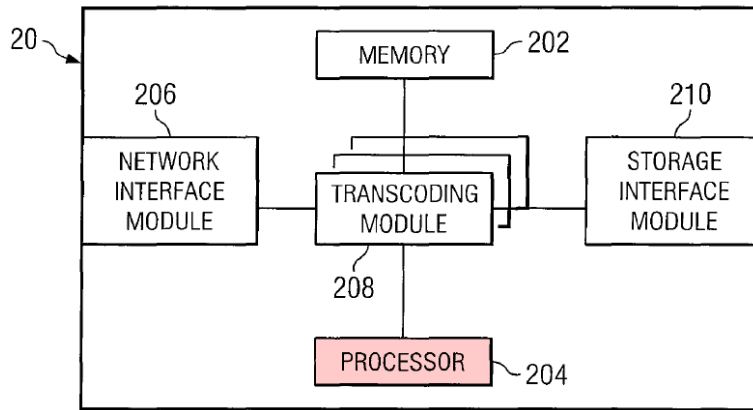


FIG. 2

Mills, Fig. 2

c. 11[b]

Mills discloses/suggests 11[b]. Ex-1003, ¶¶278-79.

Mills discloses the system comprises memory (*non-transitory computer-readable storage medium*) for storing *executable computer program instructions to be executed by the processor*. Mills, ¶51; *id.*, ¶¶22-23, 25, 40, 49, 55.

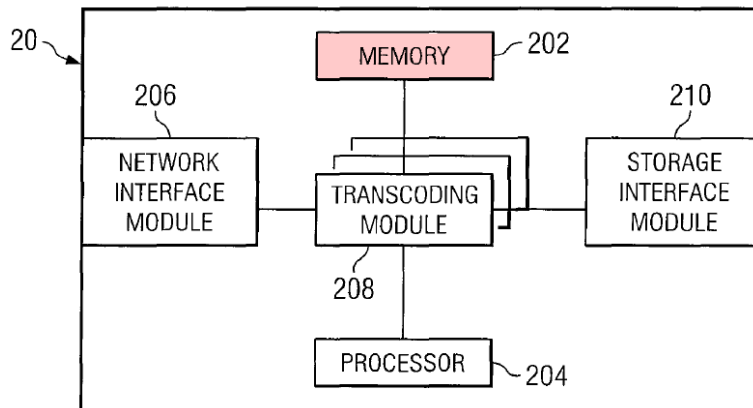


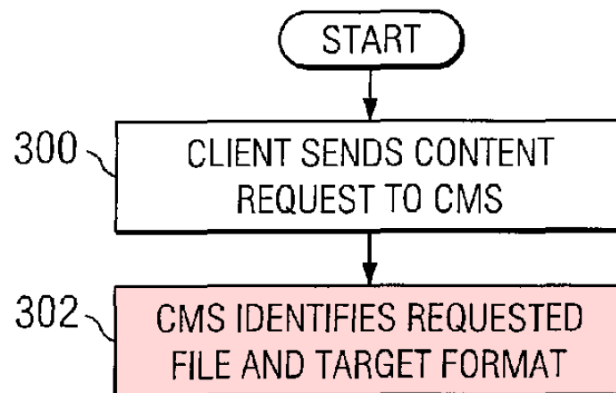
FIG. 2

The instructions would cause the processor to perform each of the functions addressed below, either alone or in combination with Mattis and/or Sackstein. Ex-1003, ¶279.

d. 1[a]/11[c]

Mills, alone or combined with Mattis, teaches/suggests 1[a]/11[c]. Ex-1003, ¶¶280-83.

Mills discloses that “clients 12 may include...computers.” Mills, ¶14; *id.*, ¶15. Mills also discloses “receiving a request identifying a media file from a first client...in a first format,” and “modifying the media file...to a second format associated with the first client” (*receiving a request to optimize an original source file from a client computing device*). Mills, ¶5; *id.*, ¶¶6, 13, 59 (“a client 12 send[s] a content request 32 for a media file 30”); Ex-1003, ¶281.



Mills, Fig. 3

The request includes “parameters for the requested media file 30” to determine “the output format and characteristics appropriate for the requested media

file 30” (*the request including a key specifying parameters to optimize the original source file*). Mills, ¶33; Ex-1001, 12:30-54; Ex-1003, ¶282.

Mattis includes object/name keys in a request to locate a specific version of a multimedia object (§VI.A.5.d (Ground 1, 1[a])), just like Mills. When implementing object/name keys and a database like Mattis’s Tag/Directory Tables in Mills, object/name keys in a request would be used to locate a specific version of a video. §VI.C.2 (motivation); Ex-1003, ¶283.

e. 1[b]/11[d]

Mills, alone or combined with Mattis, teaches/suggests 1[b]/11[d]. Ex-1003, ¶¶284-89.

Mills discloses two ways to locate a first portion of a video.

First, “load balancing server 18 may...determine at step 308 whether the requested media file 30 is available in transcoded content store 24 in the target format,” (Mills, ¶¶60-61, Fig. 3A (308)), which may be stored in portions (*id.*, ¶63) (*identifying... a first portion of an optimized version of the original source file a local cache of a video optimizer*). Ex-1003, ¶¶285-86. As explained below, Mills uses the request’s parameters for the search (*based on the key*). §§VI.C.3.f-VI.C.3.h (1[c]/11[e]-1[e]/11[g]); Ex-1003, ¶287.

Second, if the requested version is not already stored, load-balancing server initiates transcoding into the target format (Mills, ¶62), and the transcoder notifies

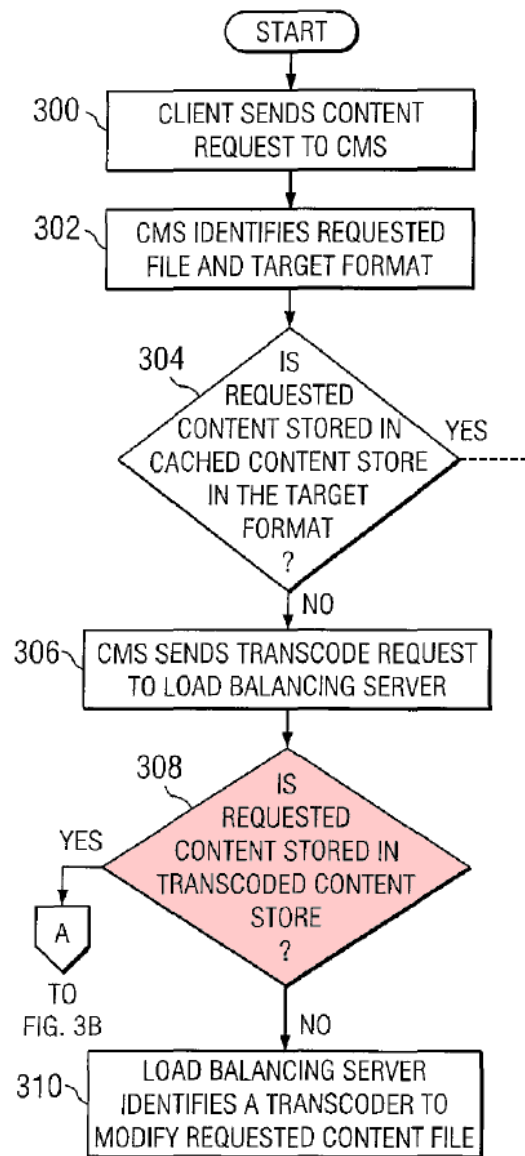
CMS when a first portion is available in the transcoded-content store (*local cache of a video optimizer*) (*id.*, ¶64) (*identifying, based on the key, a first portion*). See also *id.*, ¶¶23 (local storage), 33 (target format using parameters), 30-31, 40-41 (transcoding), Fig. 3A (318-326); Ex-1003, ¶288.

Mattis also teaches using object/name keys in requests to locate specific versions of a multimedia object (or “fragments”) (§VI.A.5.e (Ground 1, 1[b]/11[d]), just like Mills’s requests with parameters, (Ex-1003, ¶289). When implementing object/name keys and a database like Mattis’s Tag/Directory Tables in Mills, the object/name keys would be used to locate specific versions of transcoded video (or portions) in Mills’s transcoded-content stores. §VI.C.2 (motivation); Ex-1003, ¶289.

f. 1[c]/11[e]

Mills-Mattis teaches/suggests 1[c]/11[e]. Ex-1003, ¶¶290-92.

After receiving a request, Mills queries transcoded-content stores to determine whether the requested video (or portions) is already stored, or has since been generated by transcoder (*generating a query*) using the request’s parameters (*based on the key*). §VI.C.3.e (1[b]/11[d]); Mills, ¶33; Ex-1003, ¶291. Mills also explains that different versions of videos (or portions) may be stored in multiple locations, including different transcoded-content stores (*plurality of optimized versions of source files stored on a plurality of other video optimizers*). Mills, ¶¶23, 53-54; Ex-1003, ¶291.



Mills, Fig. 3A

Although Mills does not specify how searches are performed (Mills, ¶33), Mattis teaches using a request's object/name key to query the Tag/Directory Tables *(a database) maintaining a plurality of reference keys to locate a specific version of a multimedia object (or "fragments") (corresponding to a plurality of optimized versions of source files stored on a plurality of other video optimizers, each*

reference key specifying parameters of an optimized version of a source file) (§VI.A.5.f (Ground 1, 1[c])), like Mills's request parameters. Ex-1003, ¶292; When implementing object/name keys and a database like Mattis's Tag/Directory Tables in Mills, the database would be queried for keys matching a request's object/name keys to locate a specific version of video (or portions/fragments). §VI.C.2 (motivation); Ex-1003, ¶292.

g. 1[d]/11[f]

Mills-Mattis teaches or suggests 1[d]/11[f]. Ex-1003, ¶¶293-95.

Although Mills does not specify how the request's parameters are used to search the stores (Mills, ¶33; §VI.C.3.f (1[c]/11[e])), Mattis teaches receiving a response from the Tag/Directory Tables indicating whether the Tables have a matching key (*receiving, from the database, a response indicating whether the key matches a reference key*). §VI.A.5.g (Ground 1, 1[d]); Ex-1003, ¶294. When implementing object/name keys and a database like Mattis's Tag/Directory Tables in Mills, Mills's CMS or load-balancing server would receive responses indicating whether a request's key matches one in the database. §VI.C.2 (motivation); Ex-1003, ¶295.

h. 1[e]/11[g]

Mills-Mattis, with or without Sackstein, teaches/suggests 1[e]/11[g]. Ex-1003, ¶¶296-99.

Mills explains the request's parameters are used to locate a specific version of video. Mills, ¶33. Mills also explains that transcoded-content stores may store "portions of media files 30." *Id.*, ¶23. And Mills envisions storing different portions of a transcoded video in different transcoded-content stores. §VI.C.2 (motivation); Ex-1003, ¶297.

Mattis teaches that, when a request's key matches a key in the Tag/Directory Tables, a location (*address*) of the multimedia object (or "fragment") is provided. §VI.A.5.h (Ground 1, 1[e]/11[g]). When implementing object/name keys and a database like Mattis's Tag/Directory Tables in Mills, the database would issue responses with the location (*address*) of a specific version of video (or portions) in different transcoded-content stores. §VI.C.2 (motivation); Ex-1003, ¶298.

Regardless, Sackstein expressly teaches transcoding and storing different portions of a video in different transcoders. §VI.C.2 (motivation) (citing by reference Sackstein, ¶¶22, 28-35, 39-40, 47, 56-58). When implementing Mills-Mattis to transcode and store different portions of a video in different transcoders, consistent with Sackstein, the *address* received from a database like Mattis's would be the address of *another video optimizer storing a segment of the requested optimized version of the original source file*. *Id.*; Ex-1003, ¶299.

i. 1[f]/11[h]

Mills, with or without Sackstein, teaches/suggests 1[f]/11[h]. Ex-1003, ¶¶300-303.

Mills suggests storing different transcoded portions of a video in different transcoded-content stores. §VI.C.2 (motivation) (citing Mills, ¶¶23-24, 38, 53). Mills also discloses combining portions into a dynamic “created media file 30b” that allows “reading transcoded content out of the created media file 30b at the same time transcoder 24 is writing to media file 30b.” Mills, ¶24. Mills thus contemplates combining different portions of a transcoded video from different transcoded-content stores into a “created media file” for streaming (*stitching the first portion...with the segment of the optimized version of the original source file stored on the other video optimizer*). Ex-1003, ¶301.

Regardless, Sackstein expressly teaches storing different portions of a video in different transcoders, searching different transcoder caches for portions of the video, and “splicing” them together for streaming. §VI.C.2 (motivation) (citing by reference Sackstein, ¶¶22, 28-35, 39-40, 47, 56-58). When transcoding and storing different portions of a video on different transcoders consistent with Mills and/or Sackstein (§VI.C.3.h (1[e]/11[g])), the portions would be collected from the different transcoders and combined/spliced together for streaming. §VI.C.2 (motivation); Ex-1003, ¶¶302-03.

j. 1[g]/11[i]

Mills, with or without Sackstein, teaches/suggests 1[g]/11[i]. Ex-1003, ¶¶304-05.

Mills streams portions of the transcoded (“created”) media file for “playback by clients” (*streaming the stitched optimized version...for playback on the client computing device*). Mills, ¶29; *id.*, ¶¶28, 33, 43, 65, 69. Sackstein teaches the same. Sackstein, ¶¶33 (playback), 40-41 (splicing for streaming); §VI.C.2 (motivation); Ex-1003, ¶305.

4. Claims 2/12

a. 2[a]/12[a]

Mills, with or without Sackstein, teaches/suggests 2[a]/12[a]. Ex-1003, ¶¶306-10.

Mills, with or without Sackstein, suggests combining different portions of a transcoded video from different transcoders into a “transcoded media file.” §VI.C.3.i (1[f]/11[h]); Ex-1003, ¶307. A POSITA would have understood that, to form the combined file, the different portions would either be copied to a single transcoded-content store *before* streaming, or the combined “created media file” generated by a transcoder/streamer would be copied to a transcoded-content store *after* streaming so all portions would be stored together (*storing a segment...in the cache of the video optimizer*) for responding to subsequent requests for the same transcoded video. Ex-

1003, ¶307 (citing Ex-1033, ¶85). Indeed, Mills discloses after transcoding/delivery to the client is complete, the entire modified media file may be stored to fulfill subsequent requests. Mills, ¶66; Ex-1003, ¶308. Although Mills stores it in the cached-content store, a POSITA would have also been motivated to store it in a transcoded-content store instead as an obvious design choice. Ex-1003, ¶¶309-10.⁷

b. 2[b]/12[b]

Mills-Mattis, with or without Sackstein, teaches/suggests 2[b]/12[b]. Ex-1003, ¶¶311-13.

Mills discloses that, after transcoding a portion of video, “transcoder 20 may send transcode response...indicating that a portion of the requested media file 30 has been transcoded” and is available for streaming. Mills, ¶41. Although Mills does not specify how transcoded versions of a video are located, Mattis teaches updating Tag/Directory Tables when writing, moving, or deleting fragments of files. §VI.A.6.b (Ground 1, 2[b]/12[b]); Ex-1003, ¶¶312-13. When implementing object/name keys and a database like Mattis’s Tag/Directory Tables in Mills, the database would be updated when collecting all portions together in a “created media

⁷ If “a segment” may be stored *prior* to stitching, this claim is also met when transcoding new portions for subsequent stitching. §§VI.C.9-VI.C.10 (cls. 7-8).

file,” as well as storing the complete “created media file” after delivery. §§VI.C.4.a (2[a]/12[a]), VI.C.2 (motivation); Ex-1003, ¶313.

5. Claim 3/13

Mills-Mattis, with or without Sackstein, teaches/suggests claim 3/13. Ex-1003, ¶¶314-16.

Mills, with or without Sackstein, teaches that different portions of a video may be stored in different transcoded-content stores. §VI.C.2 (motivation). Mattis also teaches using a matching object key in Tag/Directory Tables to locate all fragments of a requested object. §VI.A.5.h (Ground 1, 1[e]/11[g]). When implementing object/name keys and a database like Mattis’s Tag/Directory Tables in Mills, a response to a matching key in the database would indicate that different fragments/portions (*segments*) of a transcoded video (*optimized version of the same original source file*) are stored in different transcoded-content stores (*a plurality of different video optimizers*). §VI.C.2 (motivation); Ex-1003, ¶¶315-16.

6. Claims 4/14

Mills-Mattis, with or without Sackstein, teaches/suggests claim 4/14. Ex-1003, ¶¶317-19.

Mills discloses that a load-balancing server or CMS is notified that a transcoder has portion(s) of a requested video, which a transcoder/streamer accesses and streams to the client. Mills, ¶¶24, 41-43, 45; §VI.C.3.i (1[f]/11[h]). Because

Mills, with or without Sackstein, stores different portions of a video in different transcoded-content stores (§VI.C.2 (motivation)), and network systems operate on request-response principles, Mills’s transcoder/streamer handling transmission would be understood to *request* different portions of a transcoded video from different transcoded-content stores to then collect and combine them into the “created media file” for streaming. Ex-1003, ¶¶318-19. Indeed, Sackstein expressly teaches this. Sackstein, ¶47 (“the controller 260 may also issue requests for the segment to the caches of other or neighboring systems”).

7. Claims 5/15

Mills-Mattis teaches/suggests claims 5/15. Ex-1003, ¶¶320-22.

Although Mills does not expressly disclose hashed keys, Mattis teaches that object keys are a hash of a file’s contents, and name keys are a hash of a file’s name—each being a *source content signature being a hash of the original source file* (either the file itself, or its name). Mattis, 8:27-50, 8:64-9:15; *see* Ex-1001, 6:64-7:11); Ex-1003, ¶¶321. It would have been obvious to implement keys like Mattis’s in Mills’s systems. §VI.C.2 (motivation); Ex-1003, ¶321. Alternatively, it would have also been obvious to use a hash of the contents of the original version of a file as an alternative to Mattis’s name key. Ex-1003, ¶322 (citing Ex-1029, ¶67).

8. Claims 6/16

Mills-Mattis teaches/suggests claims 6/16. Ex-1003, ¶¶323-25.

Mills discloses storing multiple versions of a video (Mills, ¶48), and transcoding videos using different transcoders, which would thus be stored in different transcoded-content stores (*a plurality of optimization server storing an optimized version of the original source file*) (*id.*, ¶53; *id.*, ¶¶5-6, 54). Mills explains that a request's parameters are used to locate a specific version of video (*id.*, ¶33), but does not specify how. Ex-1003, ¶234.

Mattis, however, teaches the Tag/Directory Tables (*database*) comprise reference-key entries (*data entries*) associated with the locations of different versions of a requested object (or fragments), and updating Tag/Directory Table entries when the location of objects (or fragments) changes. §VI.A.10 (Ground 1, cl. 6). When implementing object/name keys and a database like Mattis's Tag/Directory Tables in Mills, the locations in the database would be those of the transcoded-content stores storing different versions of a video, consistent with Mills (*a plurality of optimization servers storing an optimized version of the original source file*). §VI.C.2 (motivation); Ex-1003, ¶325.

9. Claims 7/17

Mills, with or without Sackstein, teaches/suggests claims 7/17. Ex-1003, ¶¶326-29.

Mills and Sackstein both teach checking “whether the requested media file 30 is already stored” (Mills, ¶36; *see also* Sackstein, ¶58), and transcoding if not (Mills,

¶37; Sackstein, ¶58). Ex-1003, ¶327. Sackstein also expressly describes checking for, and otherwise transcoding, individual segments of video, and not just entire videos. Sackstein, ¶58; Ex-1003, ¶327.

Because Mills, with or without Sackstein, provides multiple transcoders capable of transcoding individual portions of a video upon request (§VI.C.3.h (1[e]/11[g])), some portions of an already-transcoded video would be stored in different transcoded-content stores, but others may be missing, and any missing portions would be transcoded and combined with existing portions to generate a complete transcoded video. §VI.C.2 (motivation); Ex-1003, ¶328 (citing Ex-1030, ¶¶42, 63, Figs. 11a-b). Mills also discloses a single transcoder may transcode multiple portions of a video (Mills, ¶23-24), and thus in some instances newly transcoded portions would be stored in the same transcoded-content store as a first portion (*storing the transcoded second segment in the local cache of the video optimizer*) (Ex-1003, ¶329).

10. Claims 8/18

Mills-Mattis, with or without Sackstein, teaches/suggests claims 8/18. §§VI.C.4.b (2[b]/12[b]), VI.C.9 (cl. 7); Ex-1003, ¶330.

11. Claims 9/19

Mills, with or without Sackstein, discloses/suggests claims 9/19. Ex-1003, ¶¶331-34.

Mills, with or without Sackstein, provides multiple video transcoders capable of transcoding individual video portions upon request. (§VI.C.3.h (1[e]/11[g])). And both Mills and Sackstein check whether transcoded portions/segments are already stored and transcoding missing portions/segments. §VI.C.9 (cl.7); Ex-1003, ¶332. Accordingly, when a portion of untranscoded video (*second segment*) falls between two transcoded portions (*first portion and third segment*), the untranscoded portion would be transcoded from the start of the second segment to the start of the third segment (*transcoding the original source file from a start of the second segment to a start of the third segment*) to generate a complete transcoded video, as was known. §§VI.C.3.h (1[e]/11[g]), VI.C.9 (cl.7); VI.C.2 (motivation); Ex-1003, ¶333 (citing Ex-1030, ¶¶42, 63, Figs. 11a-b).

It would have further been understood based on Mills and/or Sackstein that (some) already-transcoded segments would be stored in different transcoded-content stores (*third segment...that is stored by one of the plurality of other video optimizers*). §§VI.C.3.h (1[e]/11[g]), VI.C.9 (cl.7); VI.C.2 (motivation); Ex-1003, ¶334.

12. Claims 10/20

Mills-Mattis, with or without Sackstein, teaches/suggests claims 10/20. Ex-1003, ¶¶335-339.

Although Mills discloses deleting unnecessary video, Mills does not describe memory management for stored videos. Mills, ¶47; Ex-1003, ¶336.

Mattis teaches a “garbage collection” process that deletes fragments (*purges*) based on frequency of access. Mattis, 21:54-23:52. A POSITA would have been motivated to modify Mills to implement a “garbage collection” process like Mattis’s to manage available memory in Mills’s transcoders (*video optimizers*). Ex-1003, ¶¶337-38. In doing so, a POSITA would have expected that first portions/fragments of a video would be accessed more frequently than later fragments (e.g., starting but not completing playback of an entire video), and thus in many instances later fragments would be deleted from the transcoded-content store but the first portion would not. §VI.C.2 (motivation); Ex-1003, ¶¶338-39 (citing Ex-1031, ¶34). This is also consistent with Sackstein, which teaches deleting individual segments for memory management, but not all segments. Sackstein, ¶¶45, 59; Ex-1003, ¶339.

VII. CONCLUSION

IPR of the Challenged Claims is respectfully requested.

VIII. MANDATORY NOTICES

A. Real Party in Interest

Petitioner identifies themselves as real parties in interest.

B. Related Matters

To the best of Petitioner’s knowledge, the ’442 patent has been involved in the following matters:

- *Tir Technologies Limited v. Meta Platforms, Inc.*, No. 8:25-cv-2345 (C.D. Cal.);

- *Tir Technologies Limited v. Netflix, Inc.*, No. 1-26-cv-00056 (D. Del.);
- *Tir Technologies Limited v. The Walt Disney Company*, No. 8-25-cv-02652 (C.D. Cal.);
- *Tir Technologies Limited v. Comcast Cable Communications, LLC*, No. 1-25-cv-00885 (D. Del.); and
- *Tir Technologies Limited v. Amazon.com, Inc.* , No. 1-25-cv-00865 (W.D. Tex.).

C. Notice of Counsel and Service Information

LEAD COUNSEL
Jessica Kaiser (Reg. No. 58,937) kaiser-ptab@perkinscoie.com PERKINS COIE LLP 1900 Sixteenth Street, Suite 1400 Denver, CO 80202-5255 Telephone: (303) 291-2300
BACK-UP COUNSEL
Matthew A. Lembo (Reg. No. 75,633) Lembo-ptab@perkinscoie.com PERKINS COIE LLP 1155 Avenue of the Americas, 22 nd Fl. New York, NY 10036-2711 Phone: 212.262.6900 Christopher Marando (Reg No. 67,898) Marando-ptab@perkinscoie.com PERKINS COIE LLP 700 Thirteenth Street N.W. Suite 800

Washington, DC 20005-3960 Phone: 202.654.6200
--

Petitioner consents to electronic service. All services and communications to the attorneys listed above may be sent to:

Meta-Tir-IPR@perkinscoie.com

D. Power of Attorney

A power of attorney is filed herewith according to 37 C.F.R. §42.10(b).

Respectfully submitted,

/ Jessica Kaiser /

Jessica Kaiser
Reg. No. 58,937
Attorney for Petitioner

PERKINS COIE LLP
1900 Sixteenth Street, Suite 1400
Denver, CO 80202-5255

Date: May 13, 2026

CERTIFICATE OF WORD COUNT UNDER 37 CFR §42.24(d)

Pursuant to 37 C.F.R. §42.24(a), Petitioner hereby certifies that portions of the above-captioned Petition for *inter partes* review of U.S. Patent No. 10,484,442, in accordance with and reliance on the word count provided by the word-processing system used to prepare this Petition, that the number of words in this paper is 13,427. Pursuant to 37 C.F.R. §42.24(a), this word count is in compliance and excludes the table of contents, table of authorities, mandatory notices under §42.8, certificate of service, certificate of word count, appendix of exhibits, and any claim listing. This word count was prepared using Microsoft Word.

Respectfully submitted,

/ Jessica Kaiser /

Jessica Kaiser
Reg. No. 58,937
Attorney for Petitioner

Date: May 13, 2026
PERKINS COIE LLP
1900 Sixteenth Street, Suite 1400
Denver, CO 80202-5255

CERTIFICATE OF SERVICE

The undersigned hereby certifies that true copies of the Petition for *inter partes* review of U.S. Patent No. 10,484,442 and supporting materials (Exhibits and Power of Attorney) were served via overnight delivery on the Patent Owner at the correspondence address of record as listed on PAIR:

FENWICK & WEST LLP
801 CALIFORNIA STREET
MOUNTAIN VIEW, CA 94041

A courtesy copy was also sent via electronic mail to Patent Owner's litigation counsel listed below:

Scott E. Kolassa - skolassa@bclgpc.com
Nancy M. Olson - nolson@olsonstein.com
David M. Stein - dstein@olsonstein.com

Respectfully submitted,

/ Jessica Kaiser /

Jessica Kaiser
Reg. No. 58,937
Attorney for Petitioner

Date: May 13, 2026