

CCN1.x Tutorial

Nacho Solis
Glenn Scott

Feb 7, 2017



What is CCN?

Content-centric Networking (CCN)
is a
communication architecture
based on
transferring named data

The CCN communication architecture

Applications

User facing programs

Services

Base services required for network operation

APIs

Abstractions for interacting with the network

Transport

Structured and secure “end-to-end” communication

Messaging

Name-based, network wide communication using CCN messages

Framing

Transport for messages over layer 2

The CCN project

Specifications

Description of protocols and algorithms

Software

Reference software implementation

Hardware

Hardware prototypes (big and small)

Commercial community

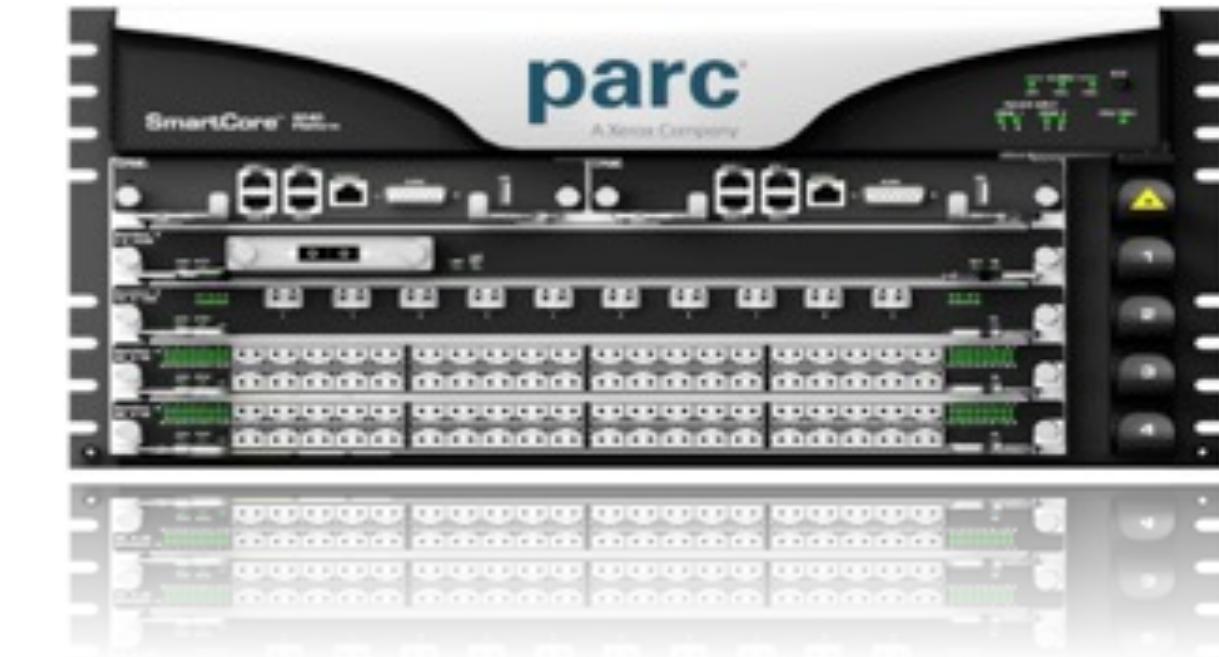
Commercial companies developing CCN

Research community

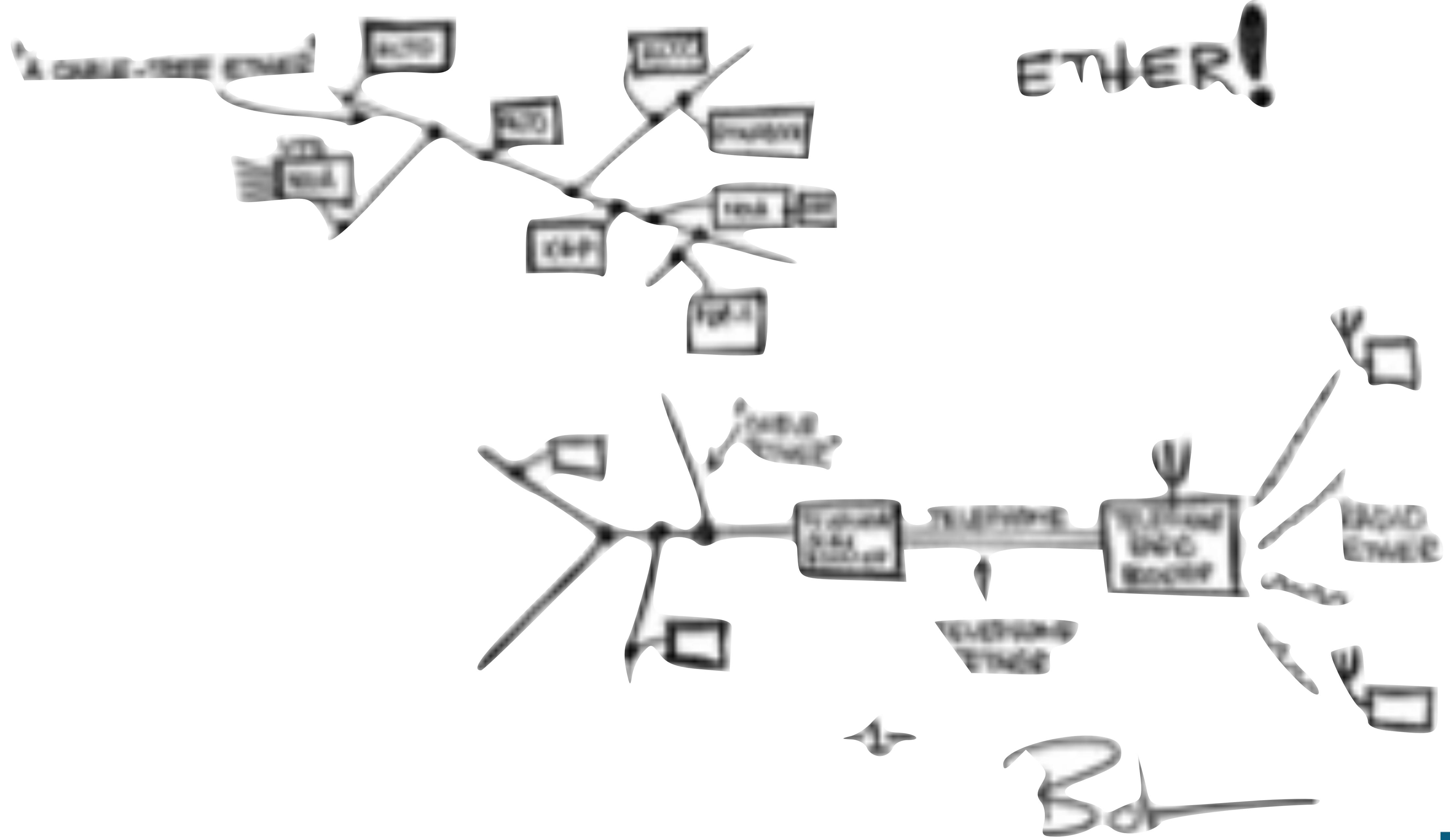
Researchers, faculty and students working on the CCN technology

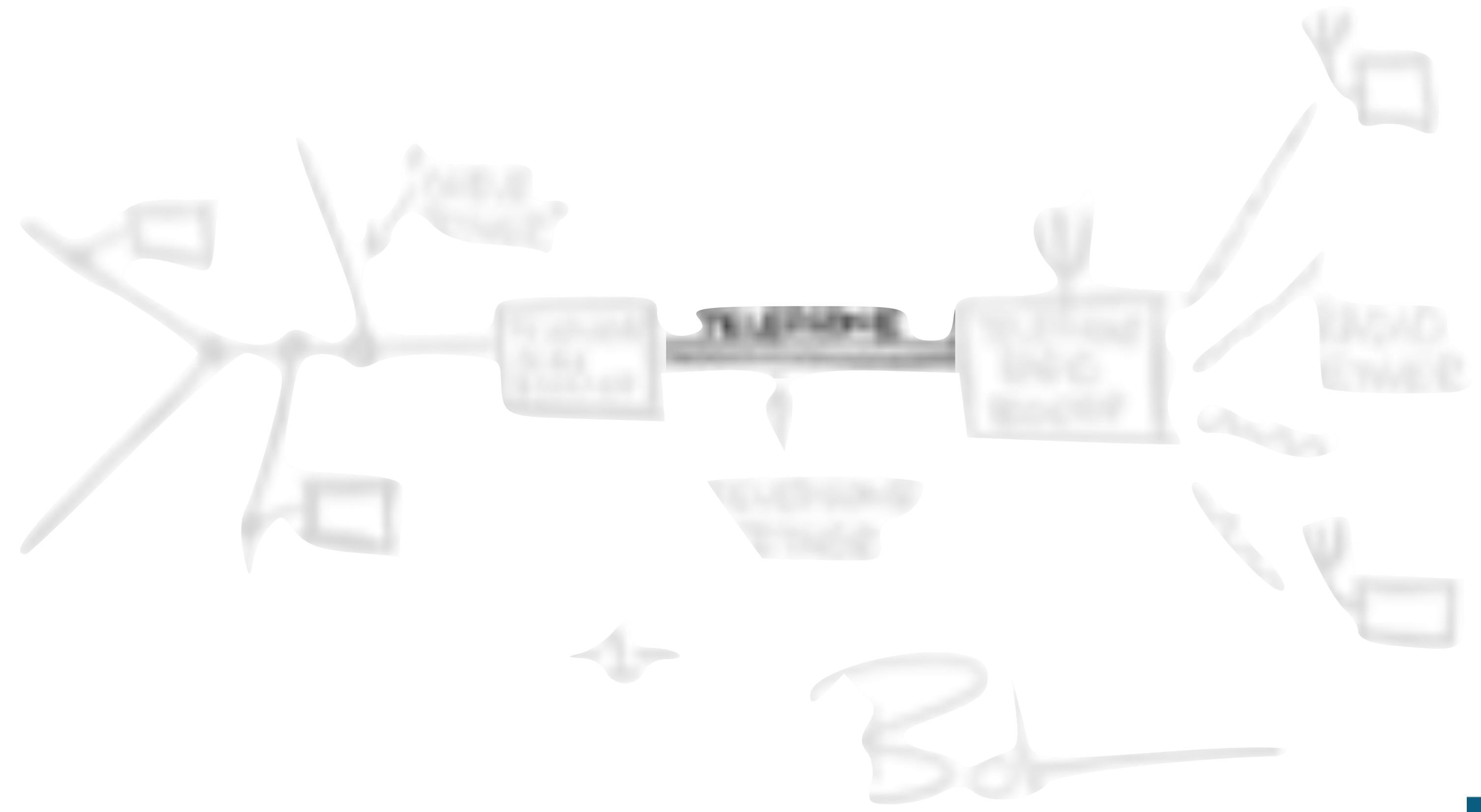
Developer community

Application developers (big and small) using CCN



History





Internet stack trouble areas

Data Center

High speed data transfers
between nearby nodes

Internet of Things

Data collection, sensing and
actuation on small devices

Web

Efficient and secure
communication channels

Enterprise

Isolation, management,
provisioning and containers

Ad-hoc / DTN

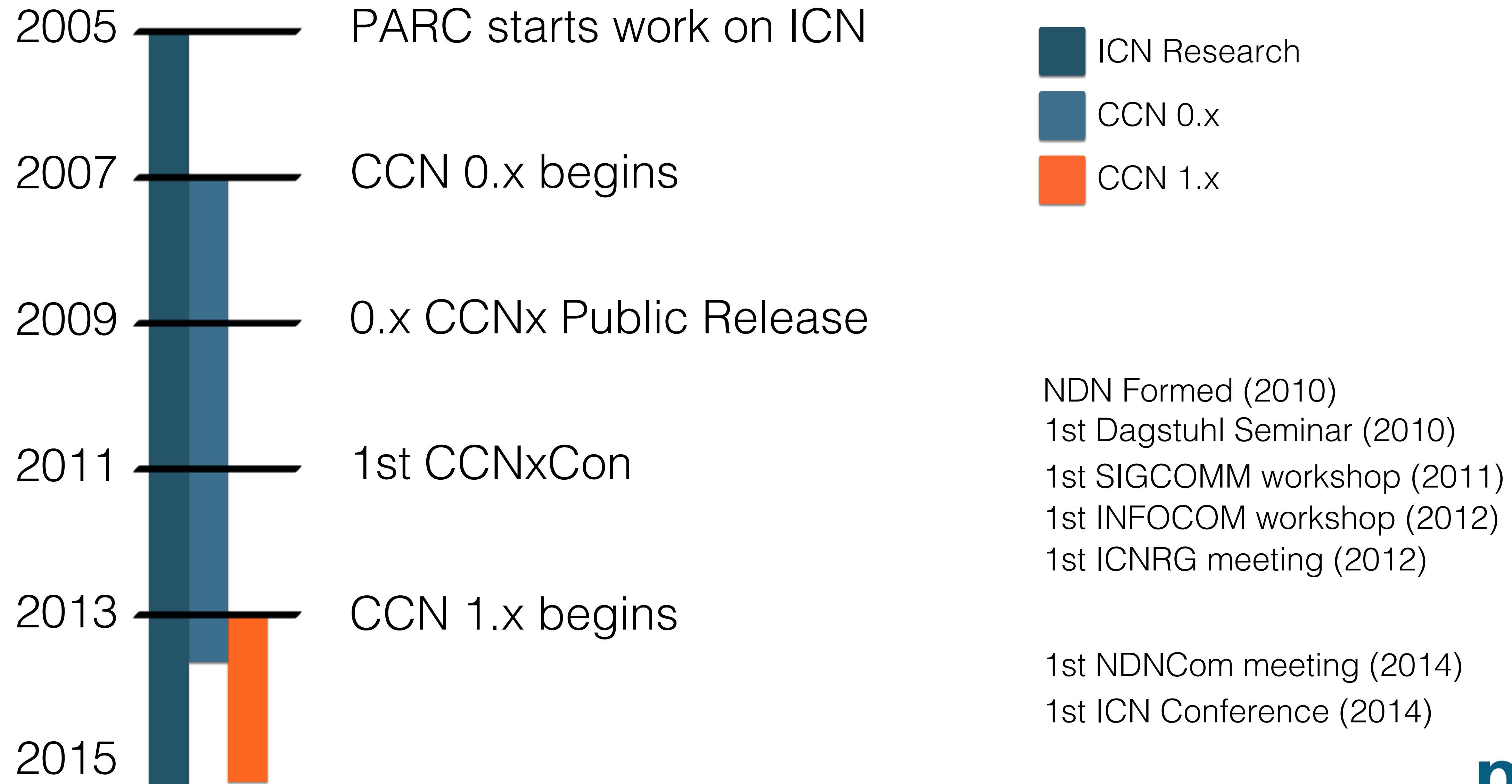
Non-traditional edge networks
with no infrastructure

De-centralized Applications

New network abstractions to
deal with communication

rethink the network
rethink the stack

Timeline



Basic concepts

CCN overview

Step 1 - Name the data

Name every piece of network data

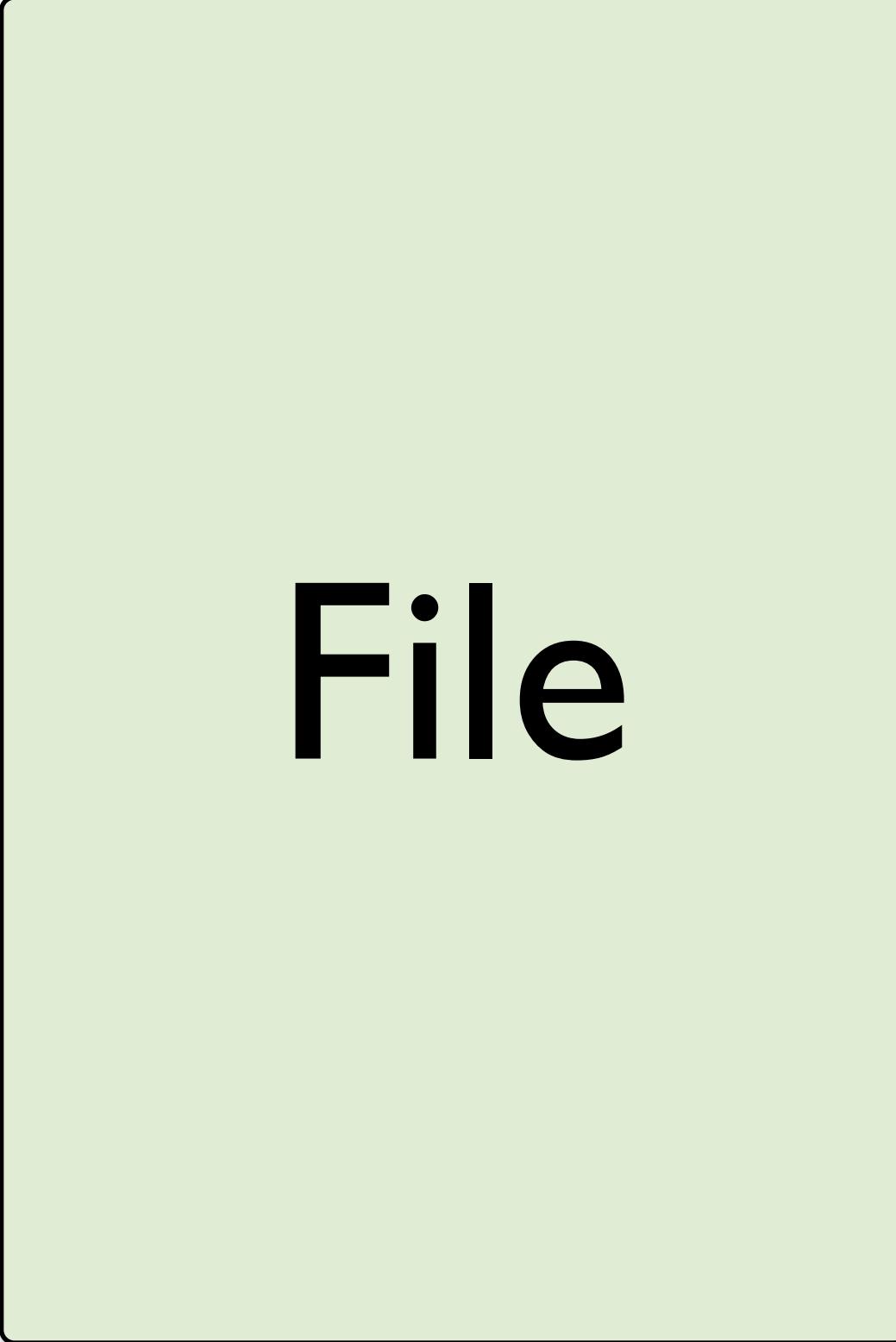
Step 2 - Secure the data

Secure every piece of network data

Step 3 - Transfer the data

Move the data to interested recipients

name data



File

Name

A large object like a file has a CCN name

File

Name

Name/chunk=1

File Chunk

Name/chunk=2

File Chunk

Name/chunk=3

File Chunk

Name/chunk=4

File Chunk

CCN also names the network sized chunks

File

Name

- Name/Manifest
-
-
- Name/chunk=1
- Name/chunk=2
- Name/chunk=3
- Name/chunk=4

Manifest

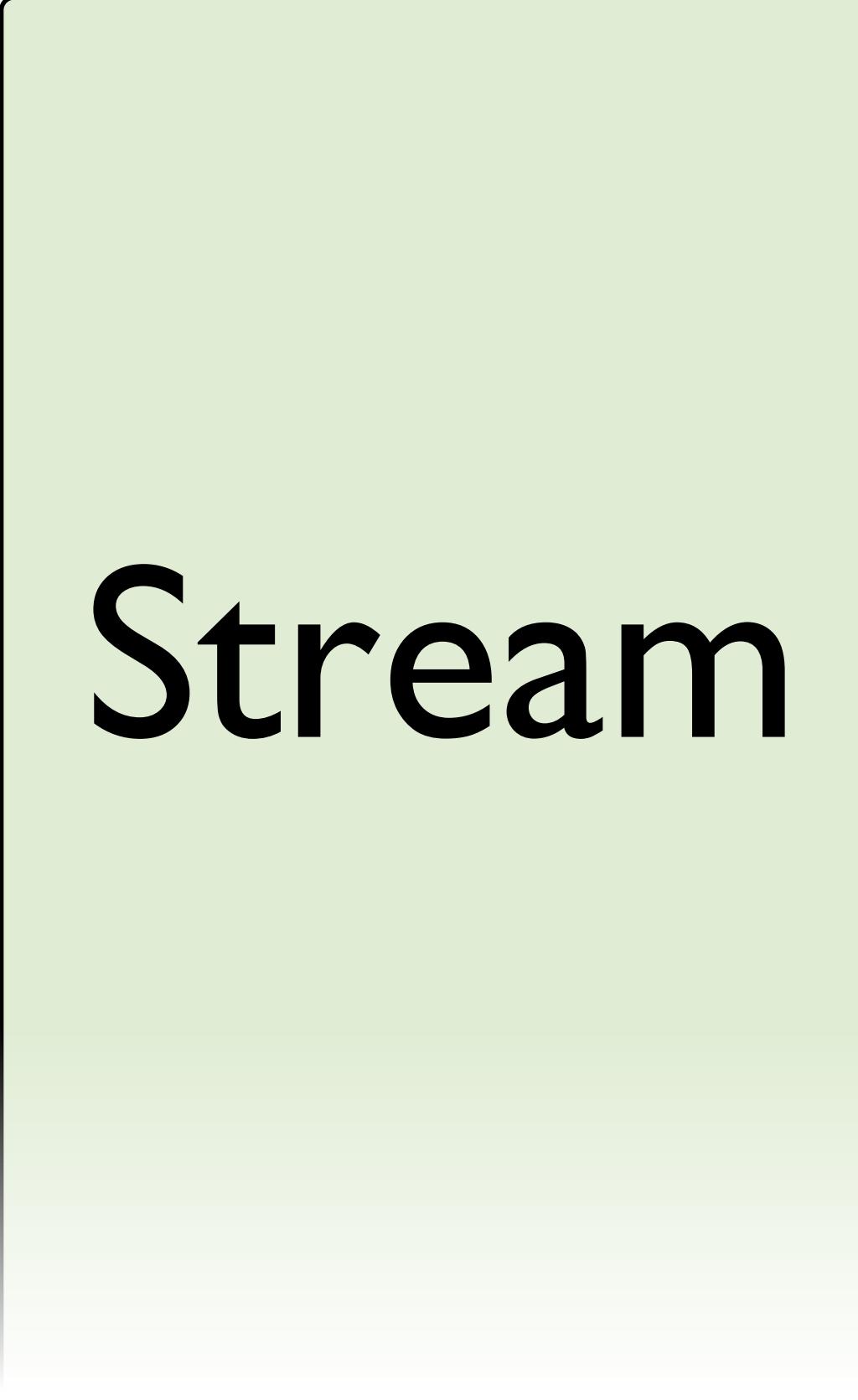
File Chunk

File Chunk

File Chunk

File Chunk

CCN creates a manifest describing the file



Stream

Name

A large object like a file has a CCN name

Stream

Name

- Name/Manifest
-
-
- Name/chunk=1
- Name/chunk=2
- Name/chunk=3
- Name/chunk=4

Manifest

Stream Chunk

Stream Chunk

Stream Chunk

Stream Chunk

The stream manifest is the metadata for the stream.

CCN name uses

Routing/Forwarding

Find the source of the data

Matching

Determine equivalence

Demultiplex

Choose between options

Structure

Inherent information

CCN name format

Name Segment based
Labeled binary segments

Hierarchical structure
Ordered sequence of segments

/seg1/seg2/seg3/seg4

CCN name example

/parc/ccnx/presentations/slides20/v=2/c=0

globally
routable
name segments

application
dependent
name segments

protocol
dependent
name segments

CCN name origins

Routable name segments

Globally coordinated namespace (like domain names)

Locally coordinated namespace (like subdomains)

Application name segments

Applications can name their data any way they want (like filenames)

Protocol name segments

Protocols use conventions in naming to transmit information (like version, chunk number, etc)

CCN name qualifier

/parc/ccnx/presentations/slides20/v=2/c=0

ContentObjectHash

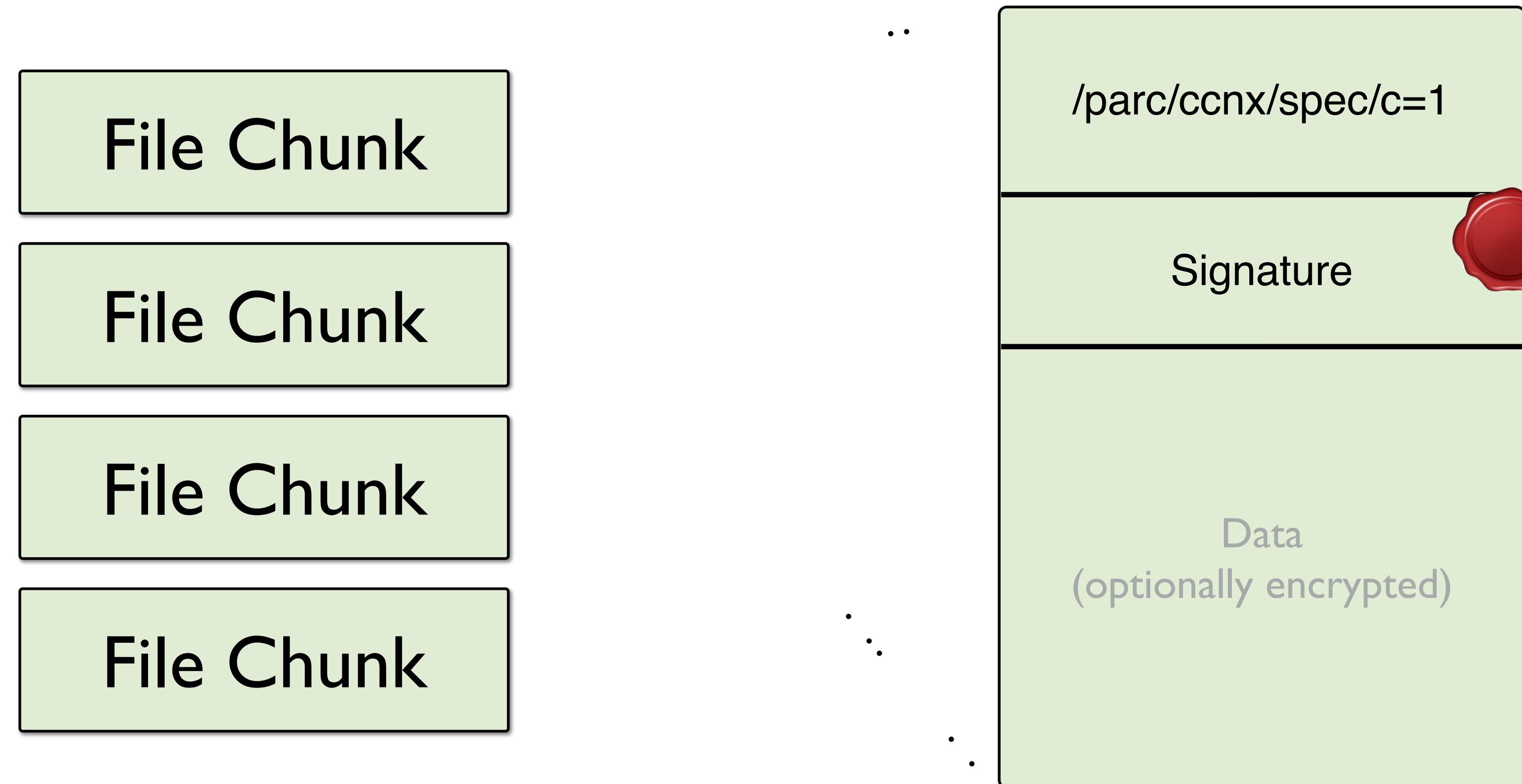
hash of the
content object
message

KeyId

identifier of key
used to sign the
object

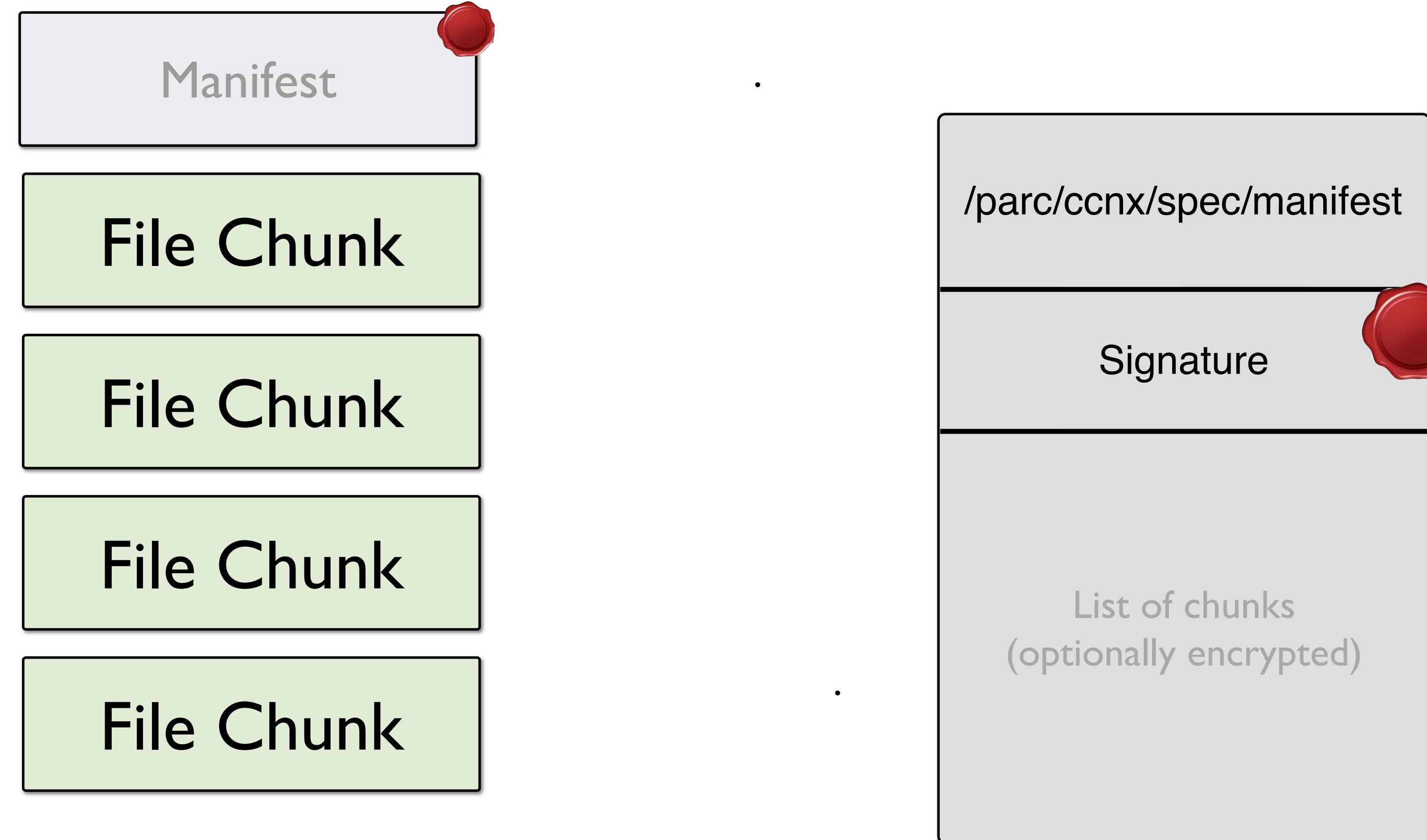
secure data

Secure single chunks



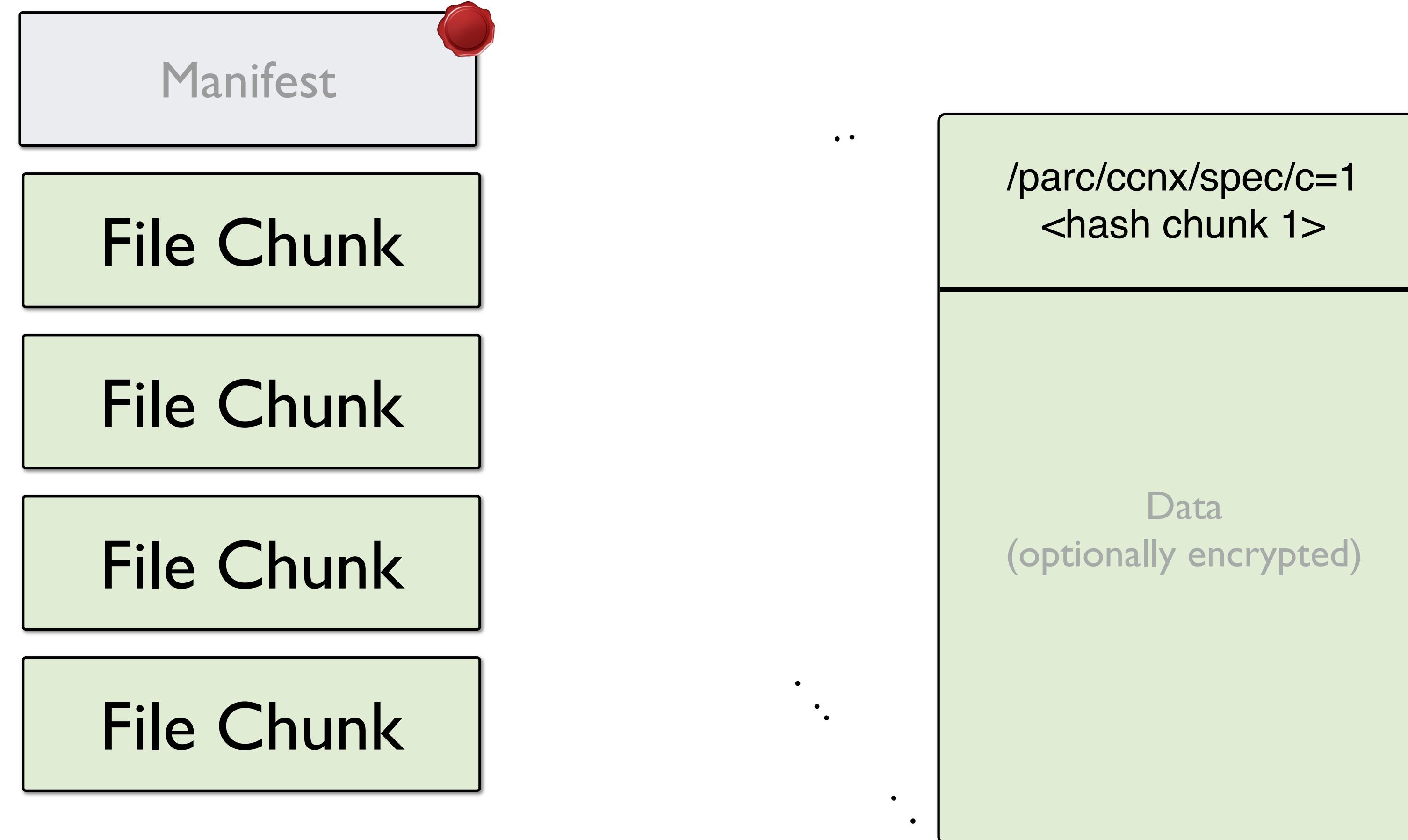
CCN names and signs every chunk

Secure whole object via manifest



CCN names and signs the file via a manifest

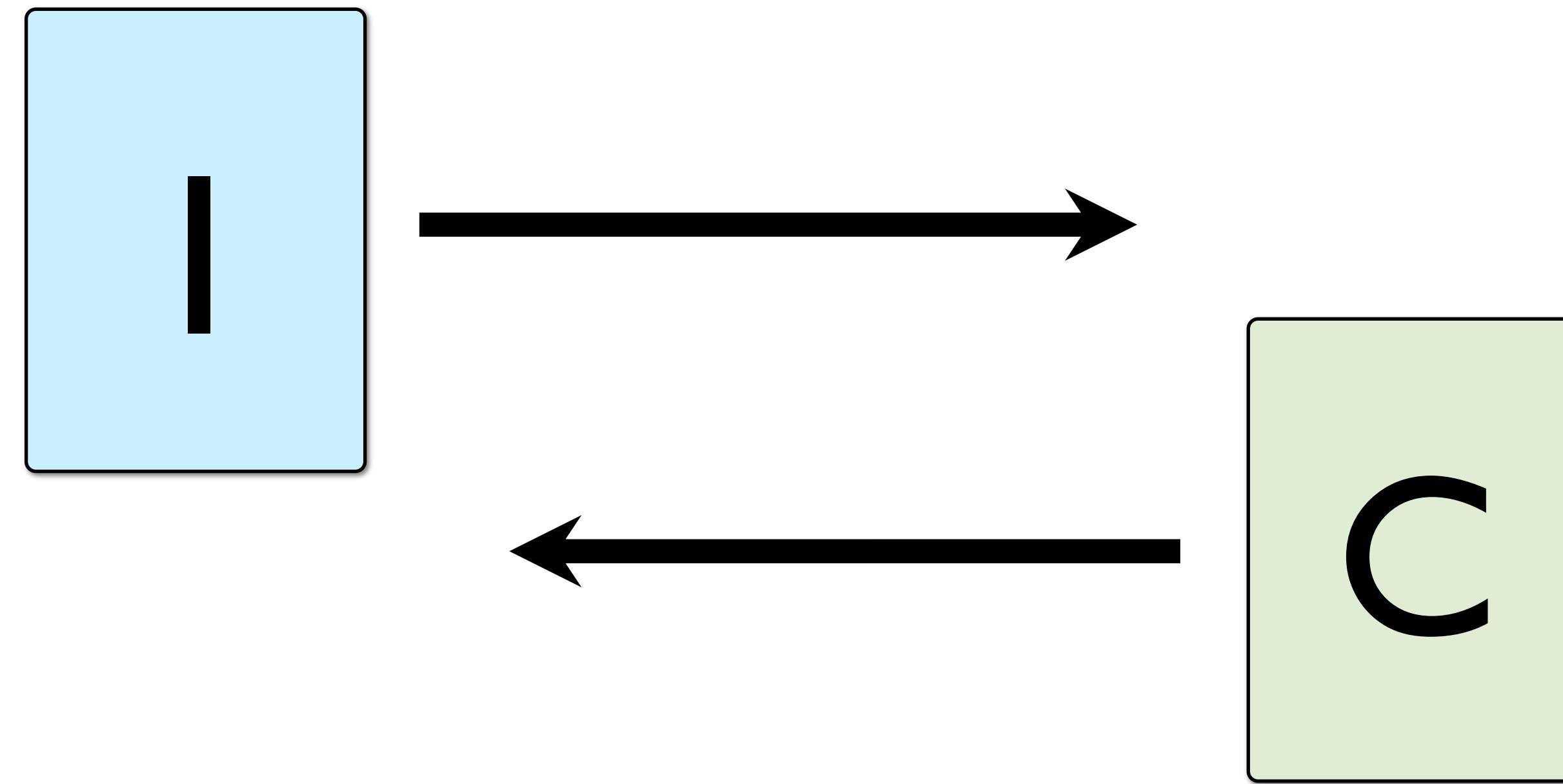
Secure via manifest



Indirectly sign every chunk through the manifest

transfer data

Core Protocol



One interest packet gets one content packet

Interest

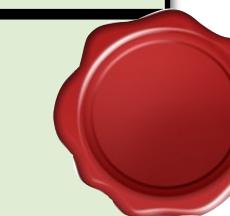
/parc/ccnx/slides1/c=5 ?

Content Object

/parc/ccnx/slides1/c=5

Signature

Data
(optionally encrypted)



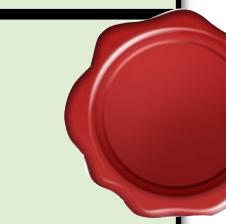
Interest

/parc/ccnx/slides1/c=5 ?

Content Object

/parc/ccnx/slides1/c=5

Signature



Data
(optionally encrypted)

Interest

/parc/ccnx/slides1/c=5 ?

KeyID=<keyId>

Content Object

/parc/ccnx/slides1/c=5

Signature by <keyId>



Data
(optionally encrypted)

Interest

/parc/ccnx/slides/c=5 ?

COHash=<hash>

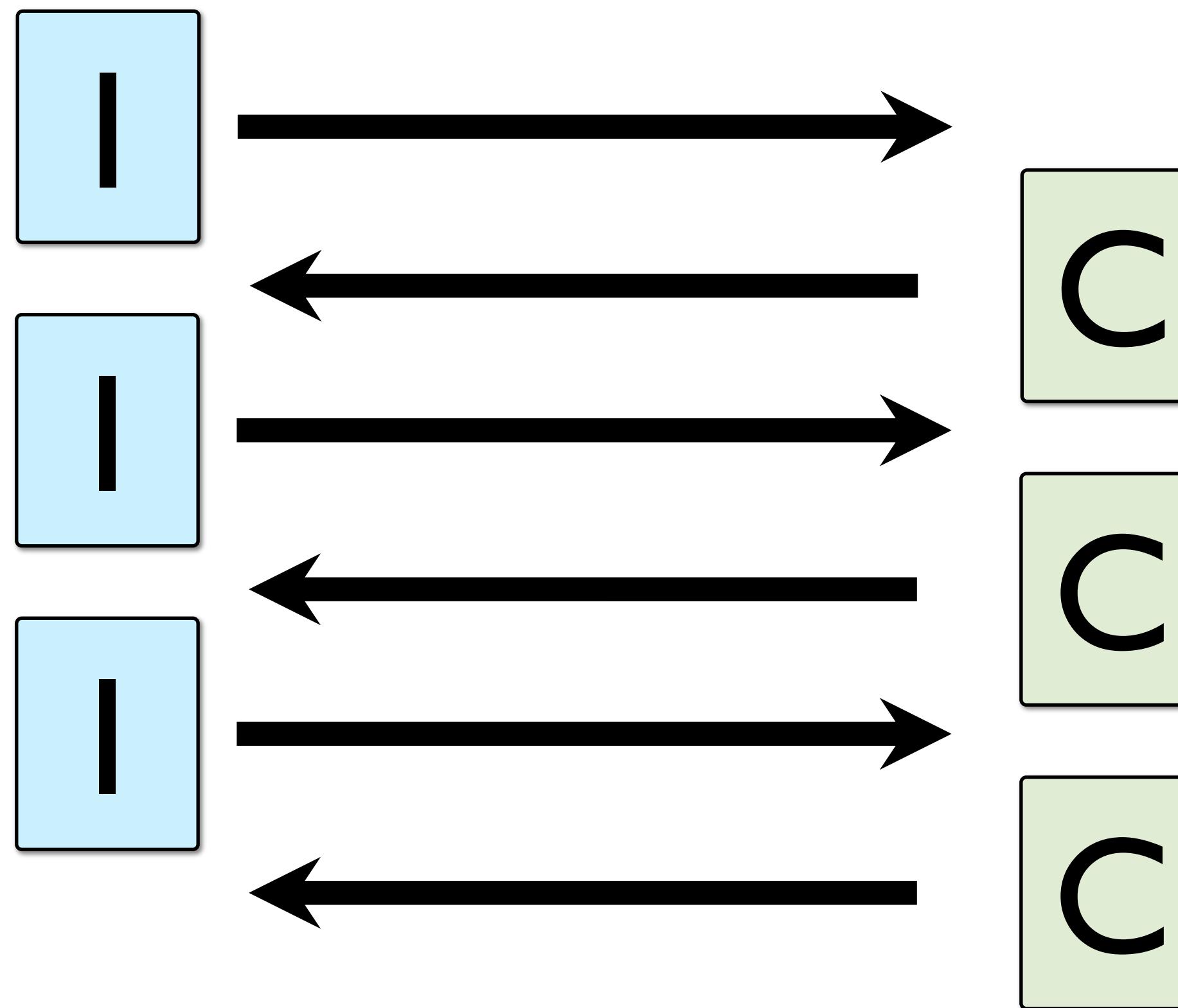
Content Object

/parc/ccnx/slides/c=5

Data
(optionally encrypted)

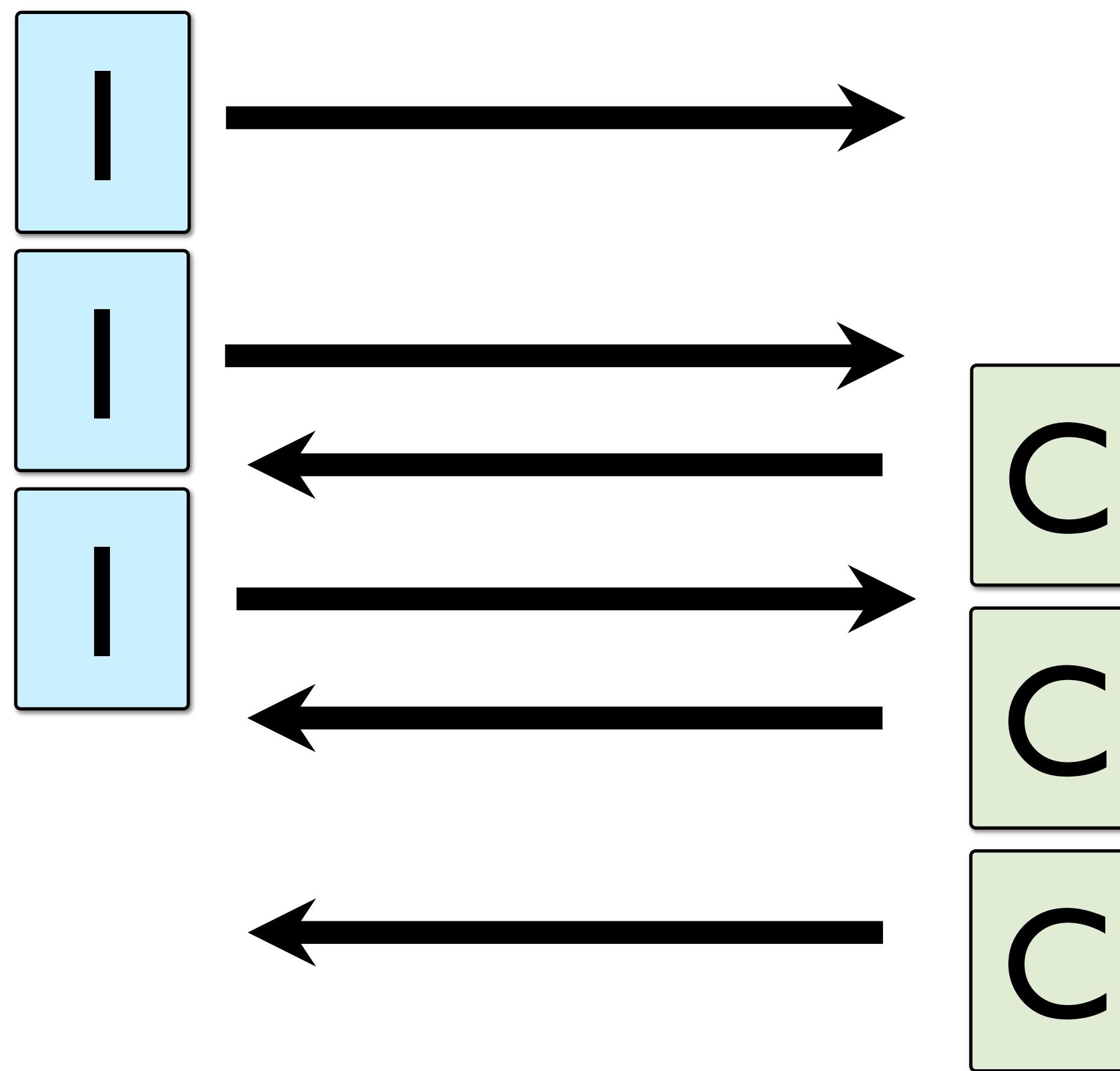
hash

Transport Protocols



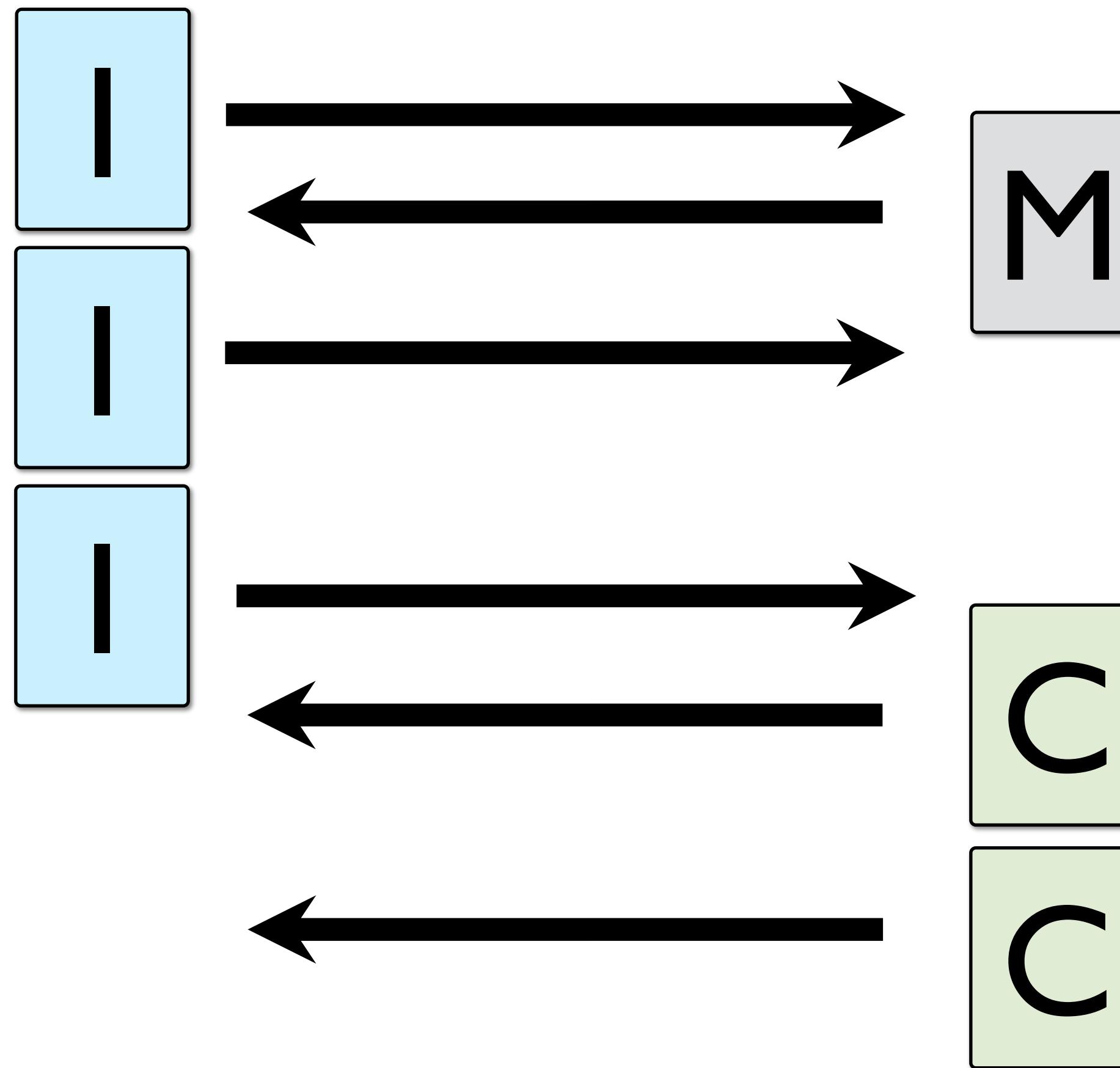
Transport protocols are built on core exchanges

Transport Protocols



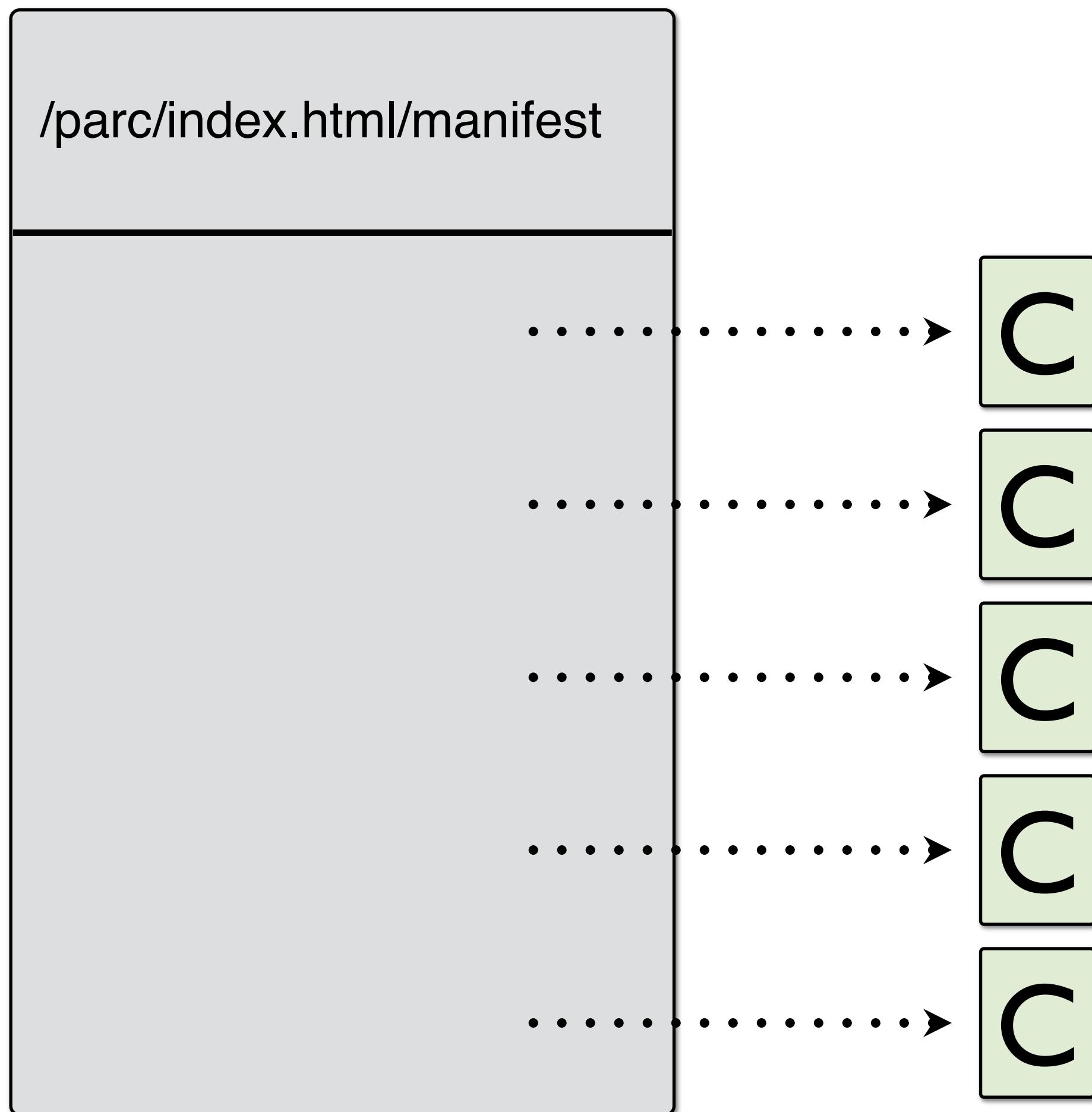
Transport protocols can do parallel requests

Transport Protocols

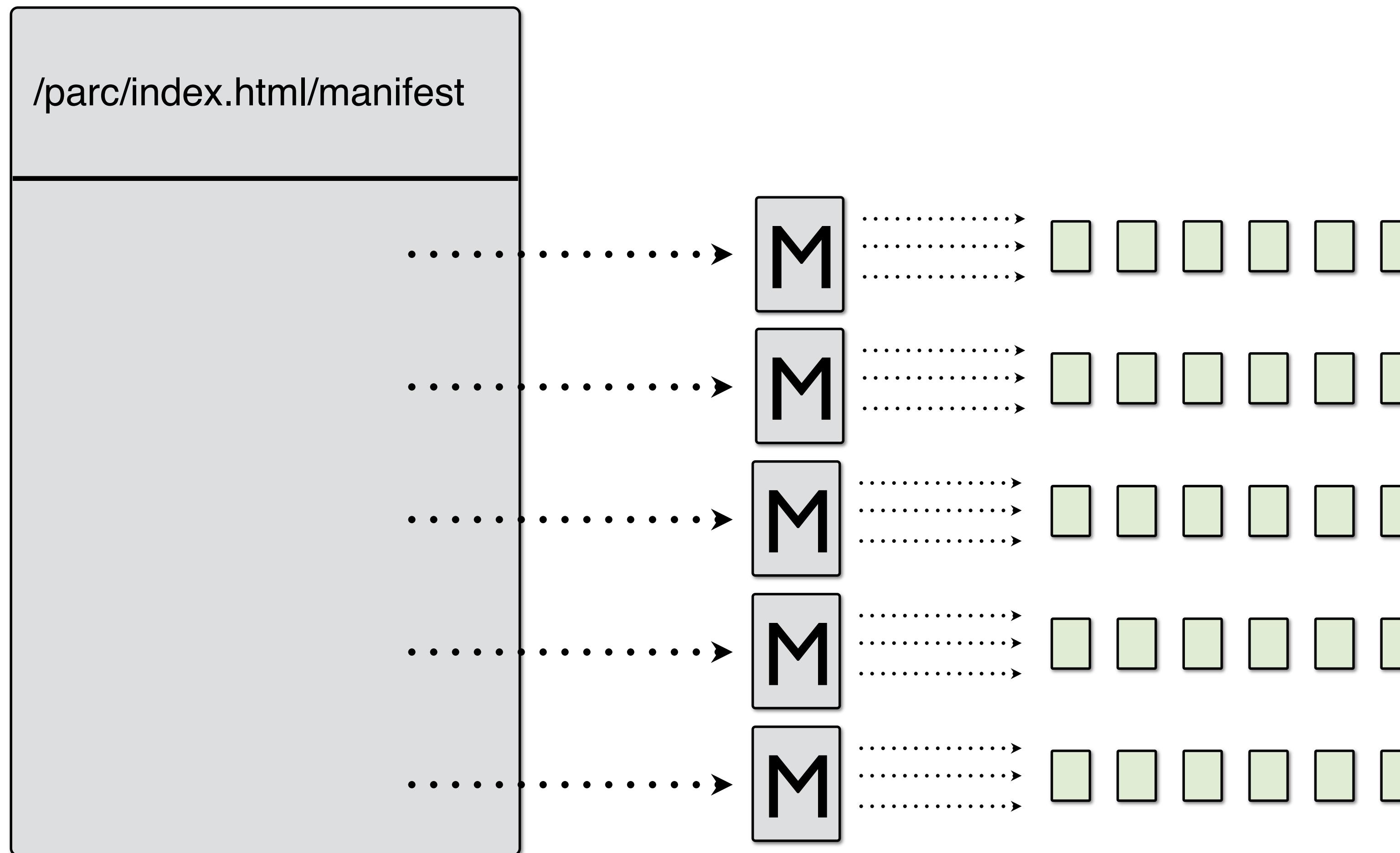


Transport protocols can use manifests

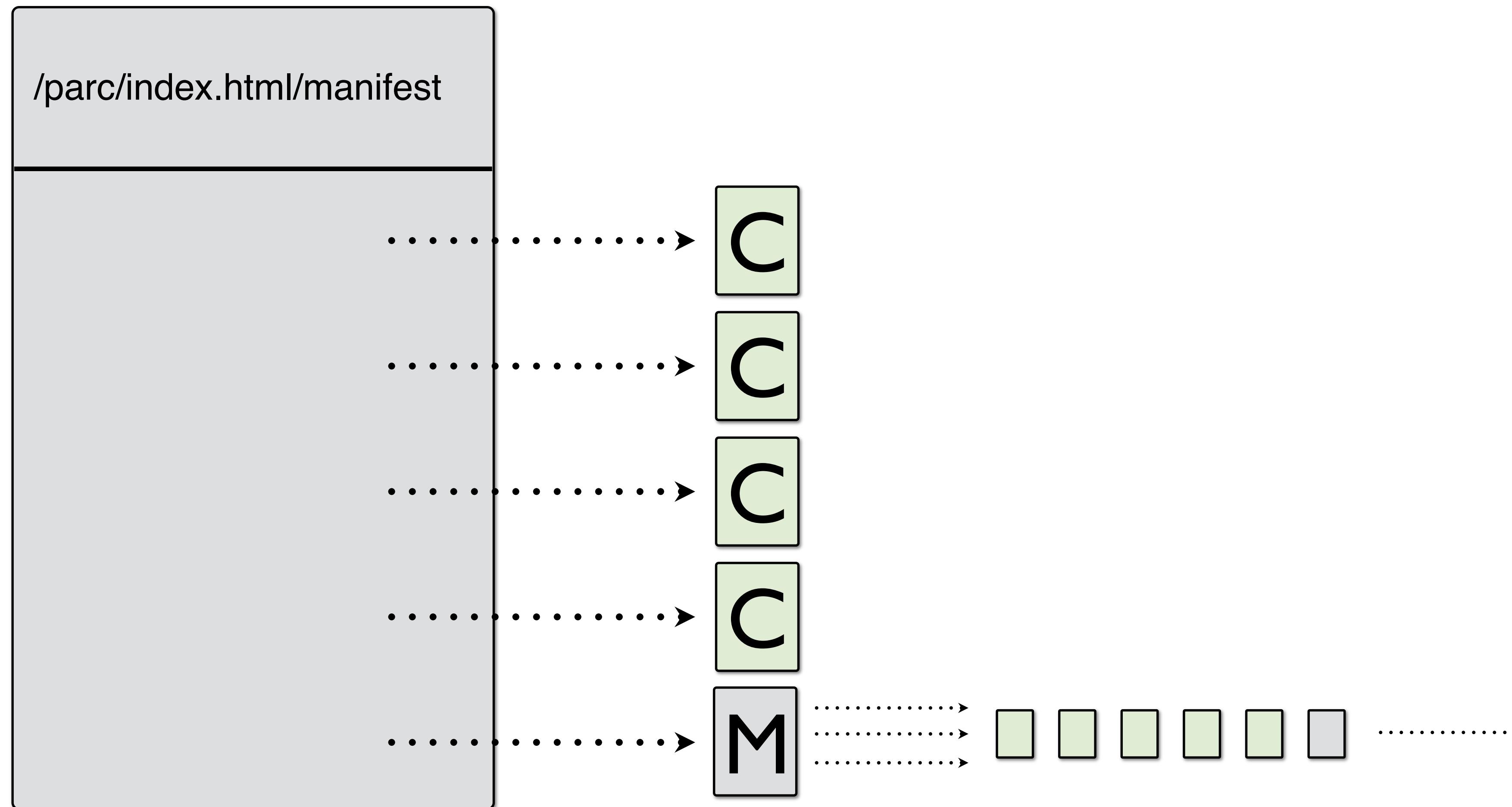
Manifest structure



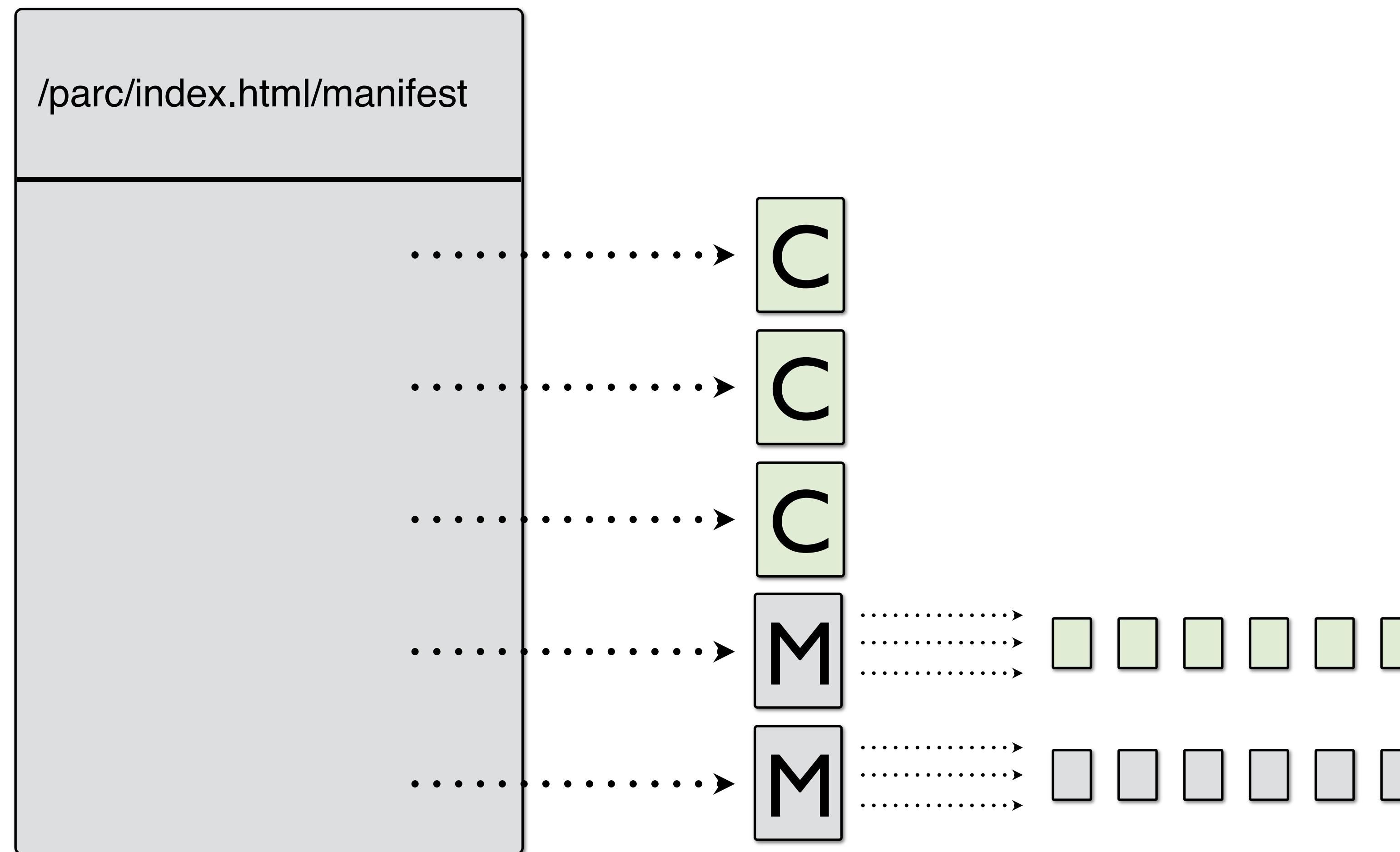
Manifest structure



Manifest structure



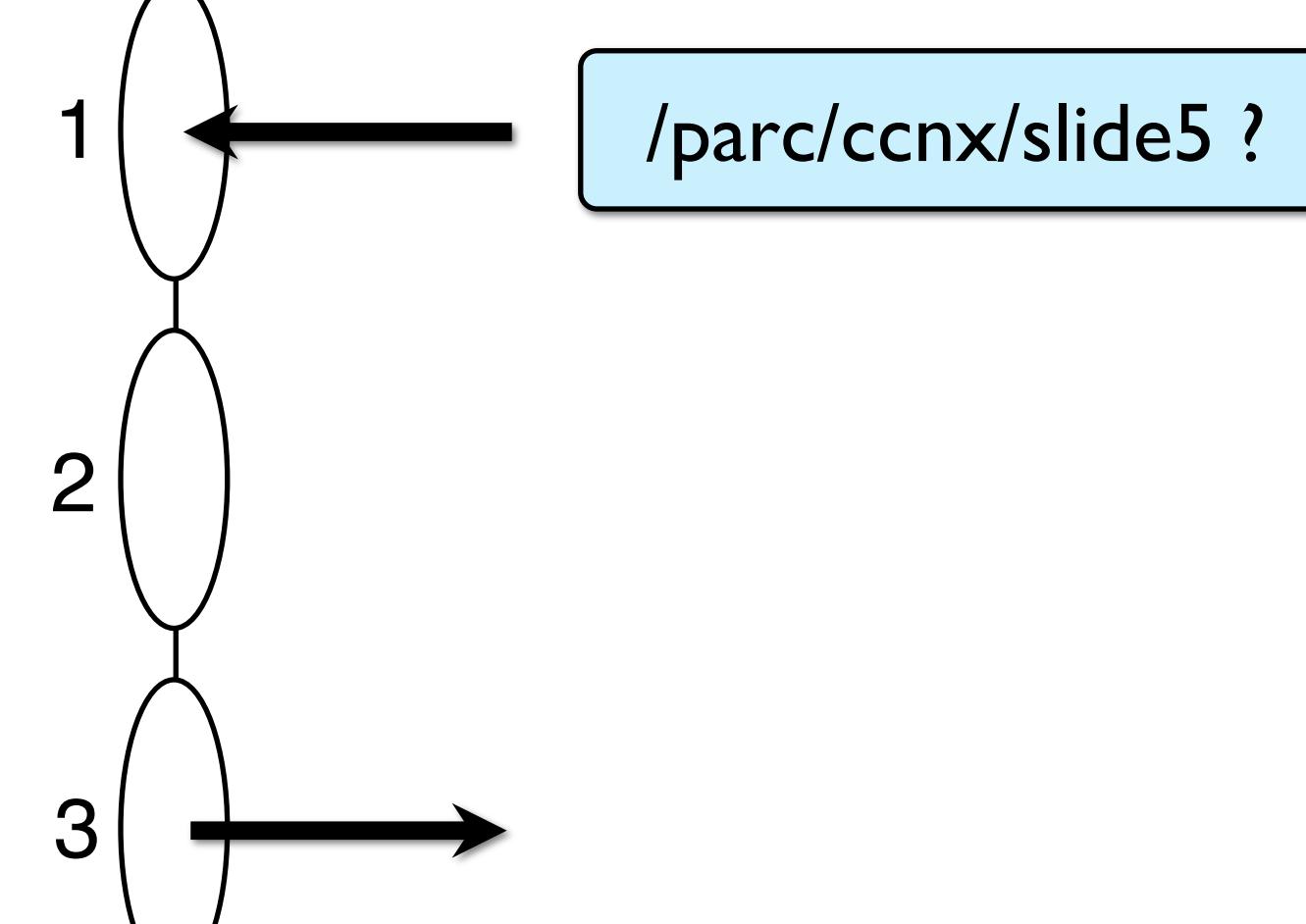
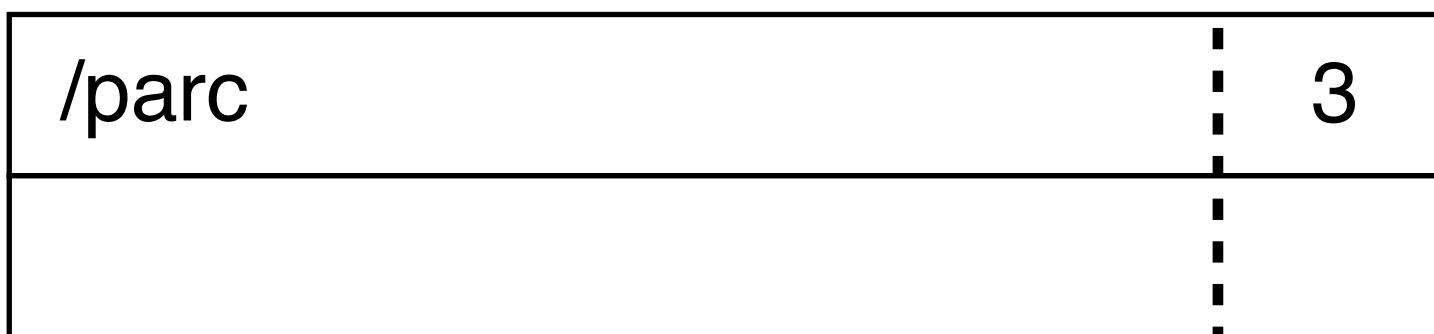
Manifest structure



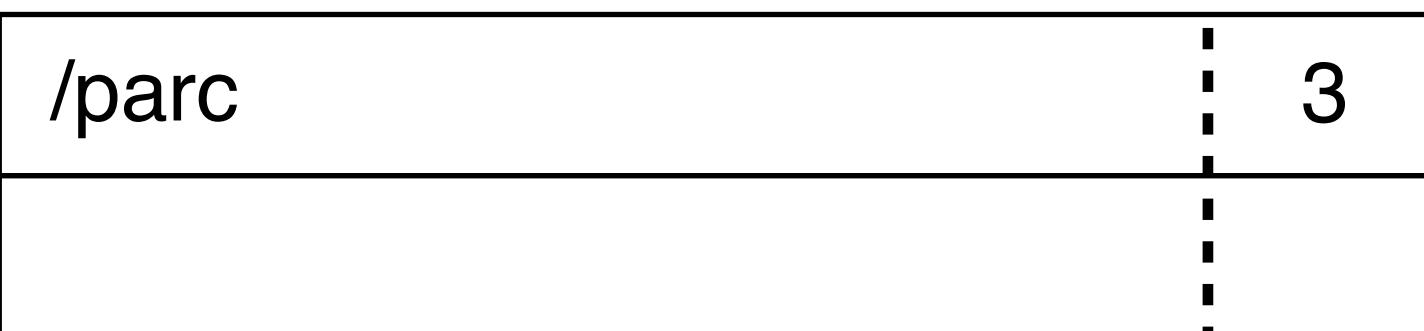
CCN

CCN - Forwarder

FIB



FIB



PIT



1 ← /parc/ccnx/slides ?

2

3 →

FIB

/parc	3

PIT

/parc/ccnx/slide5	1

1

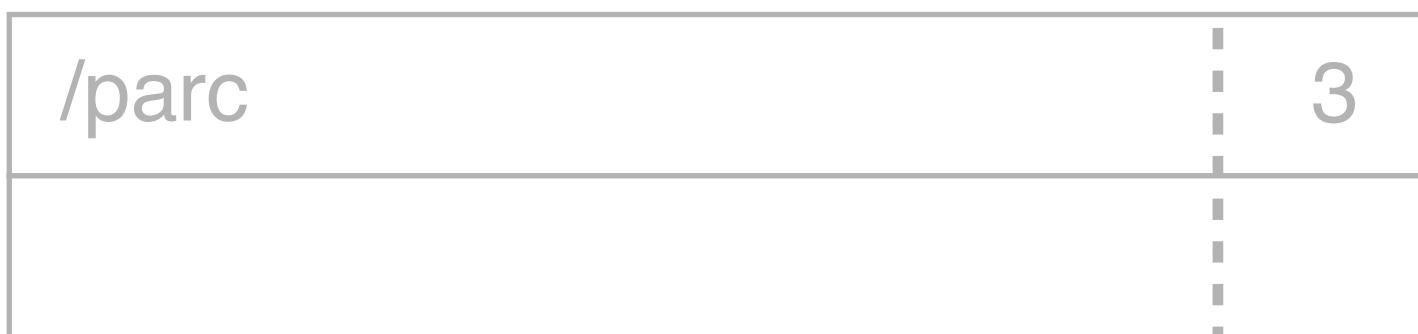
2

3

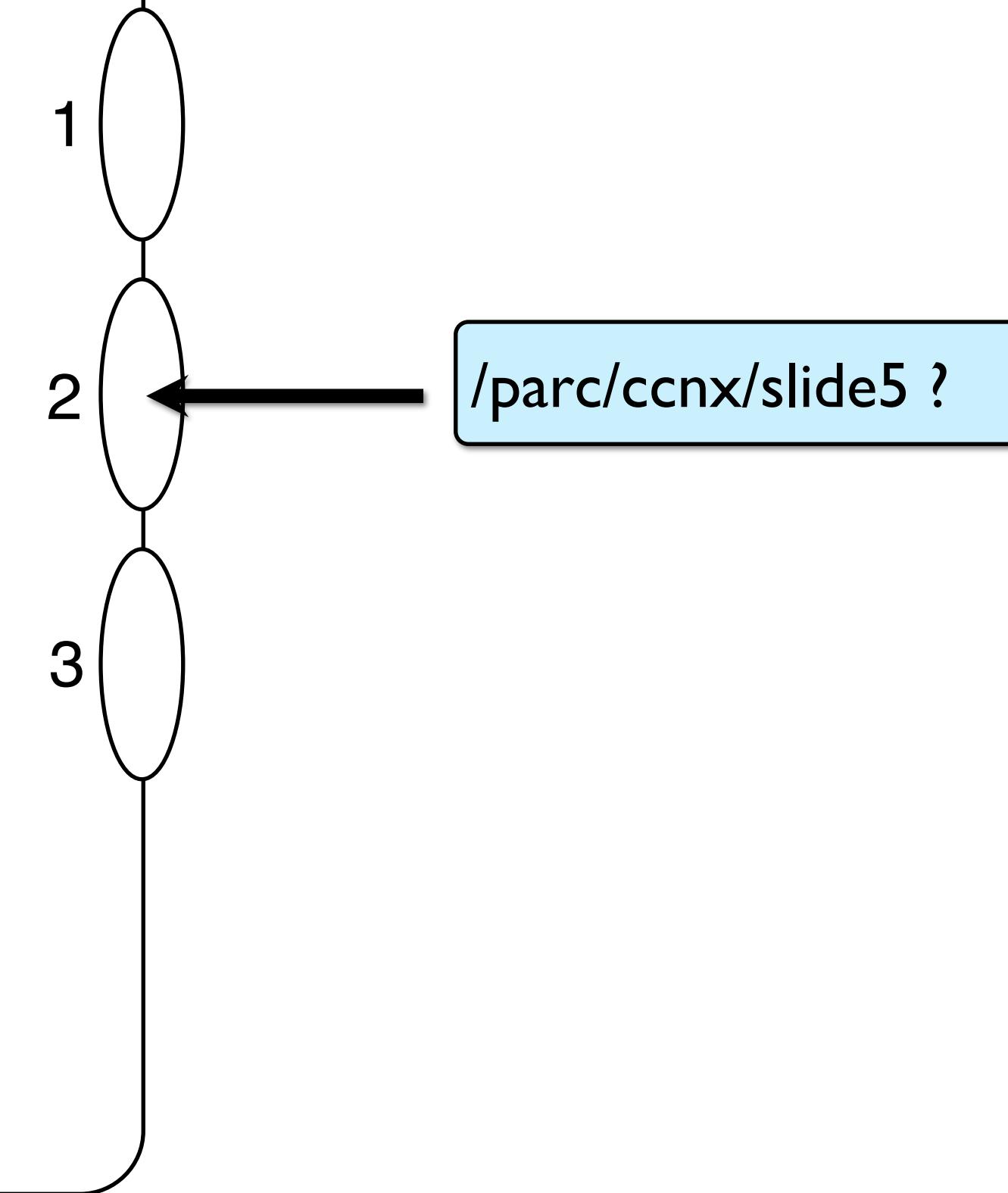
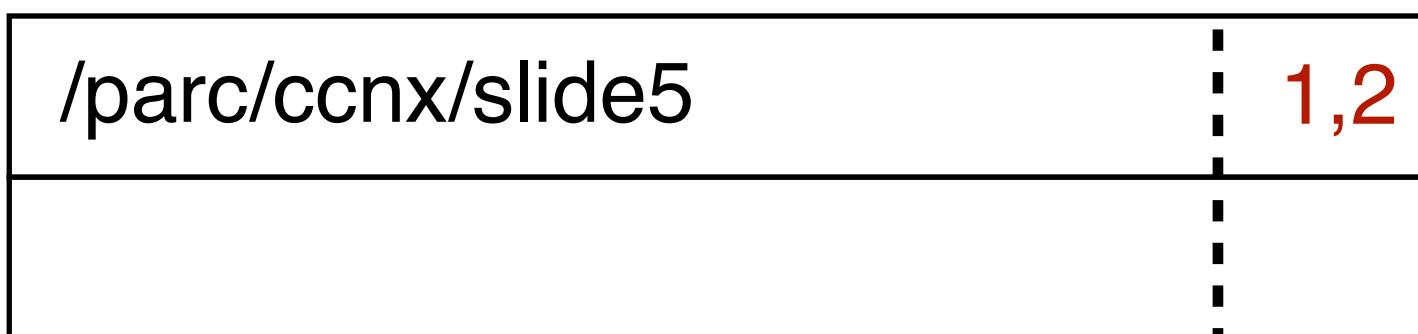
/parc/ccnx/slide5 ?

/parc/ccnx/slide5 ?

FIB



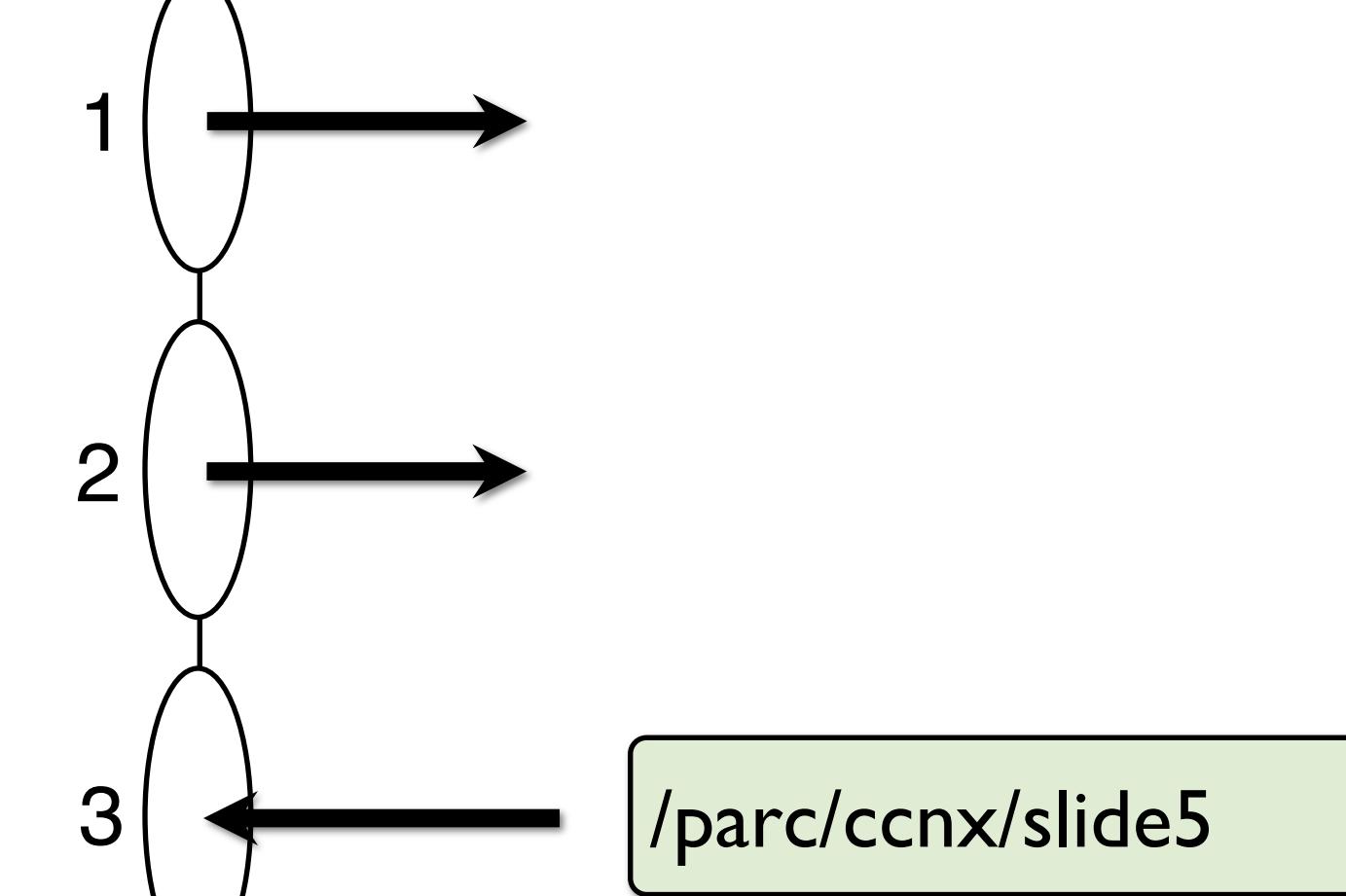
PIT

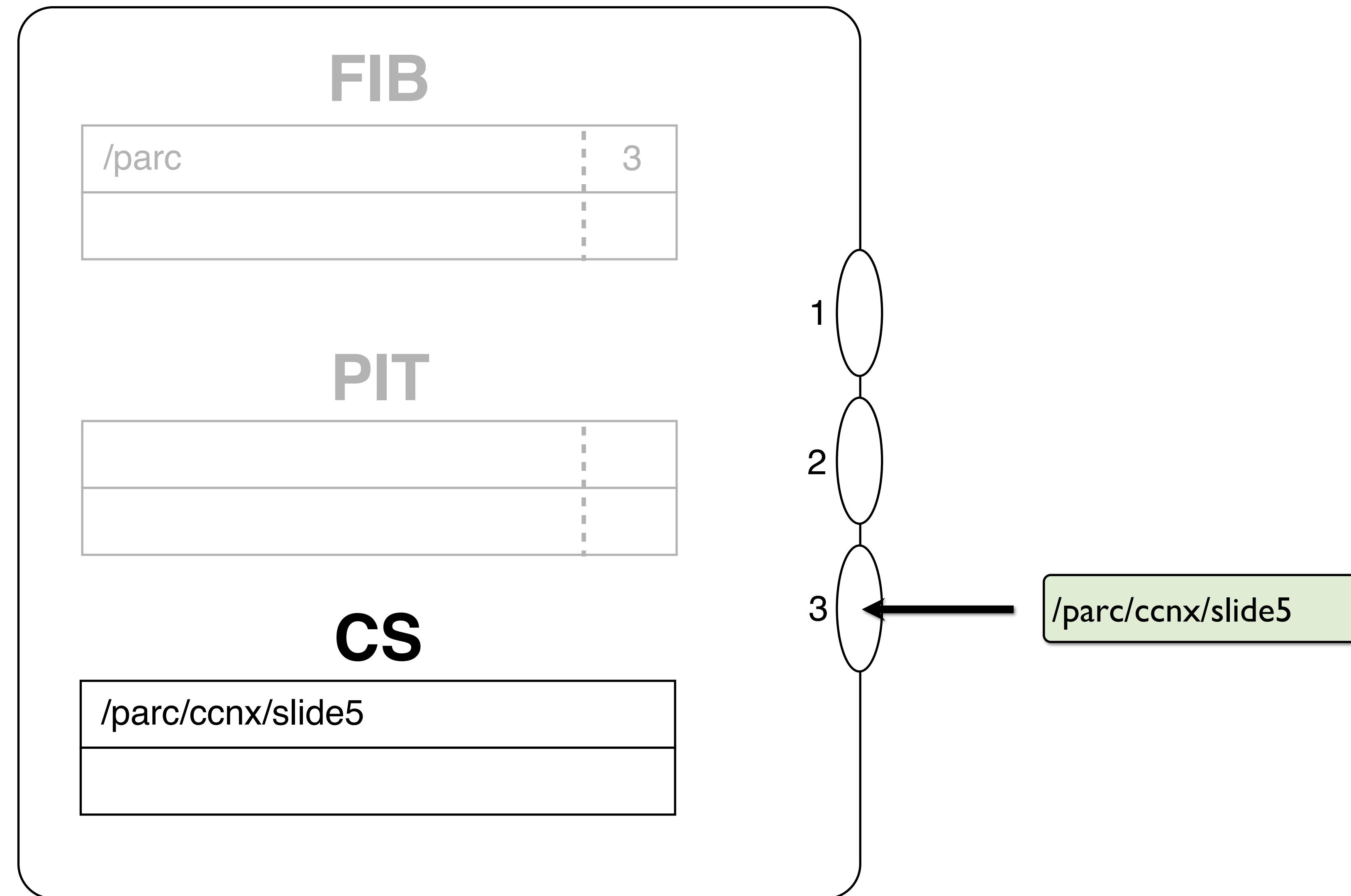


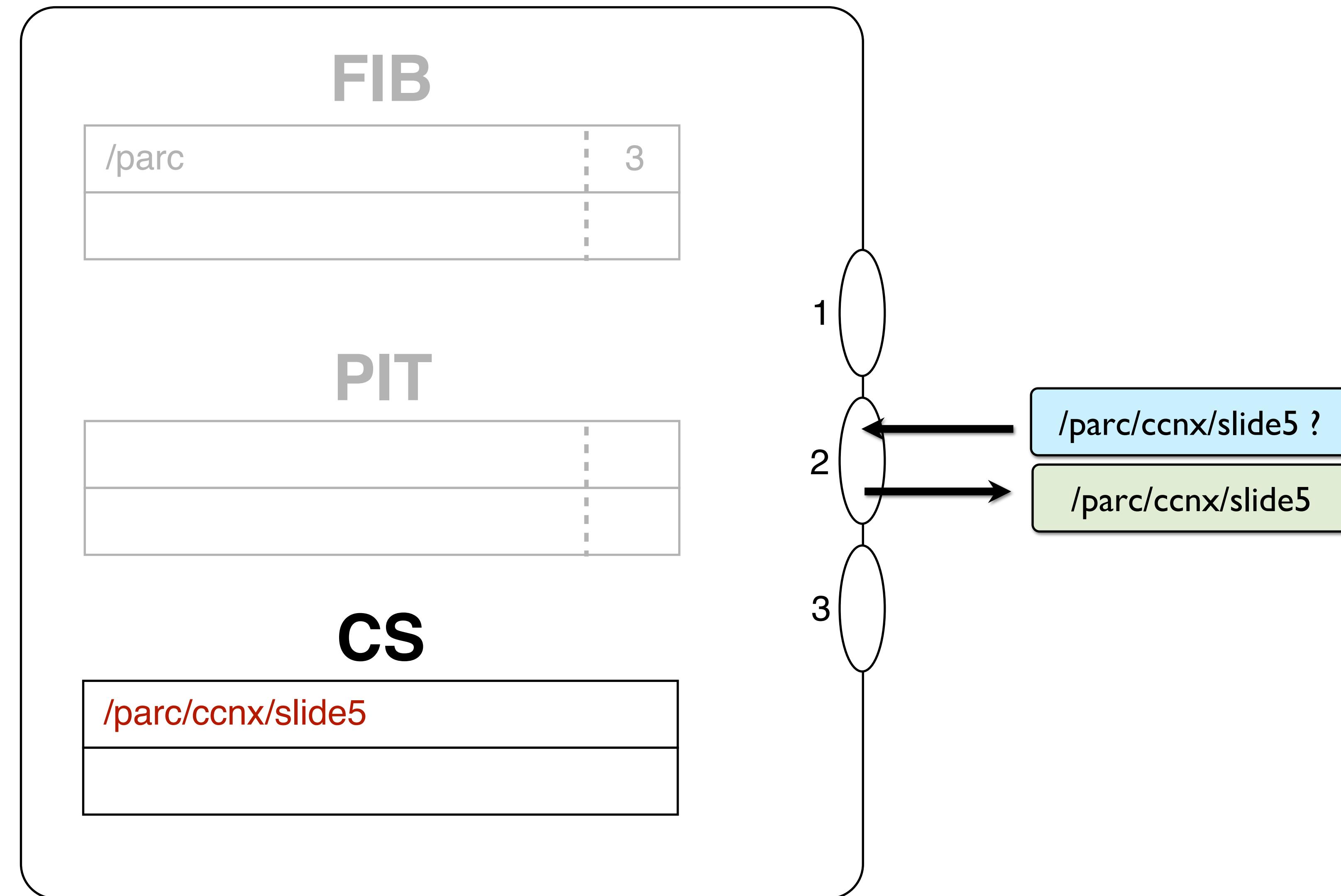
FIB



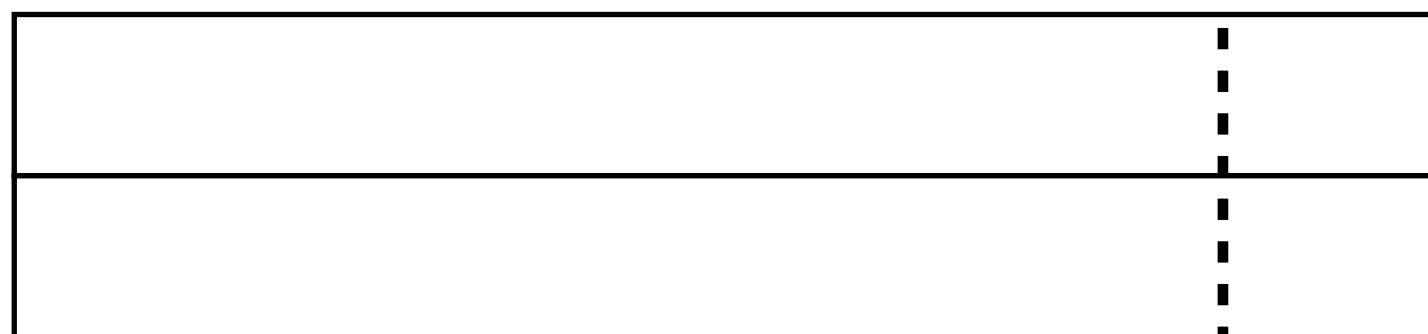
PIT





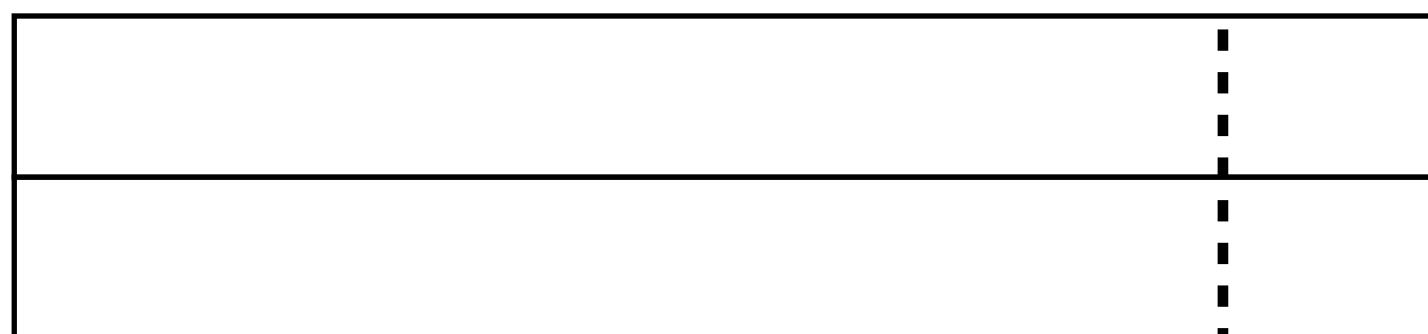


FIB



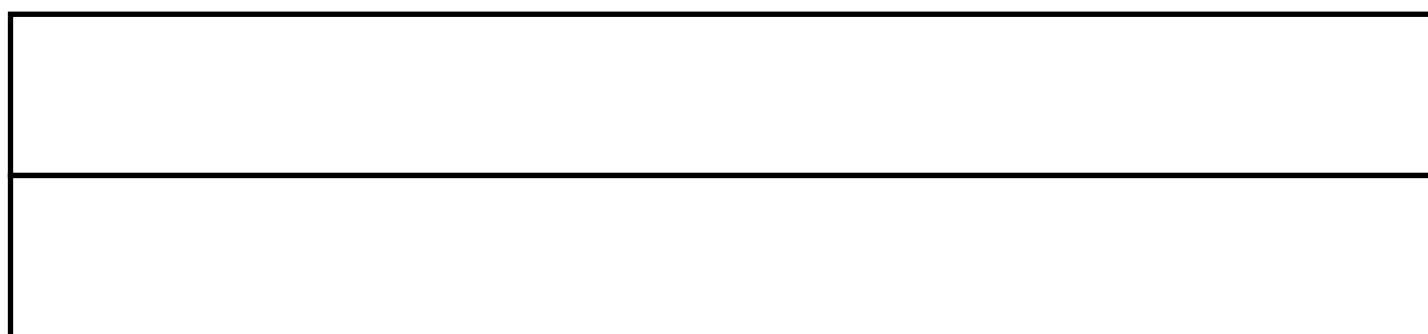
Forwarding Information Base
Store information about what face
to follow to find a given name

PIT



Pending Interest Table
Store information about what
interests are pending

CS

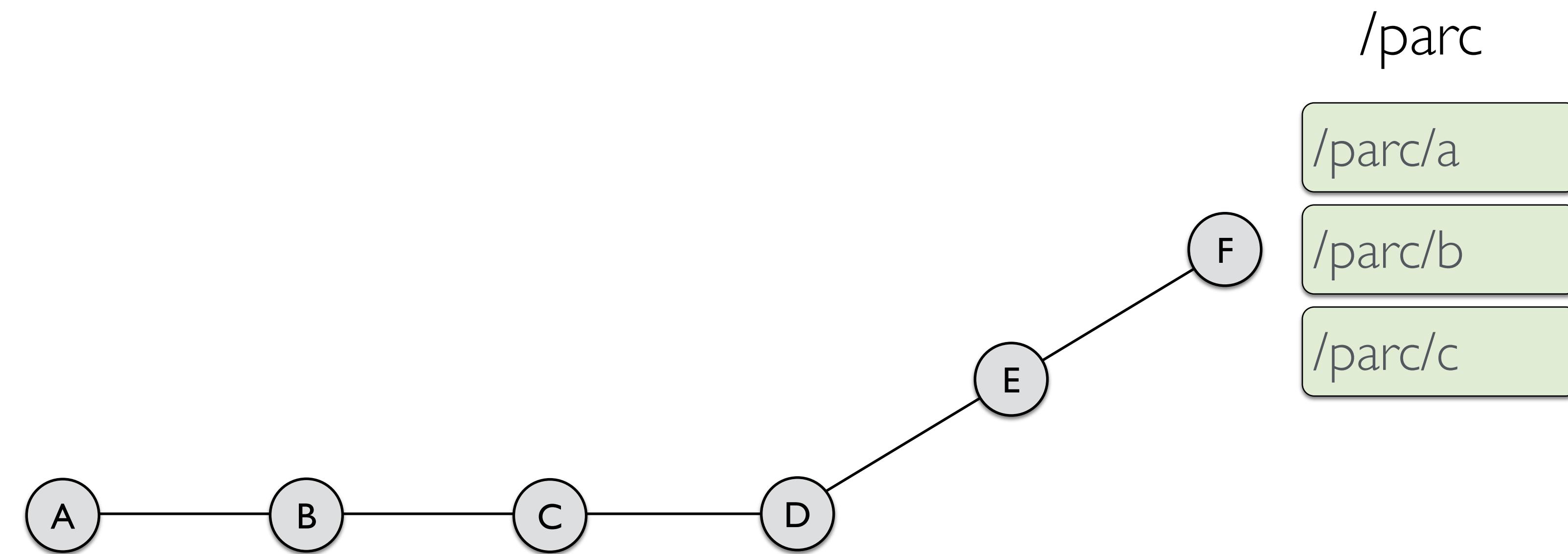


Content Store
Buffer content objects for
potential reuse

- 1
- 2
- 3

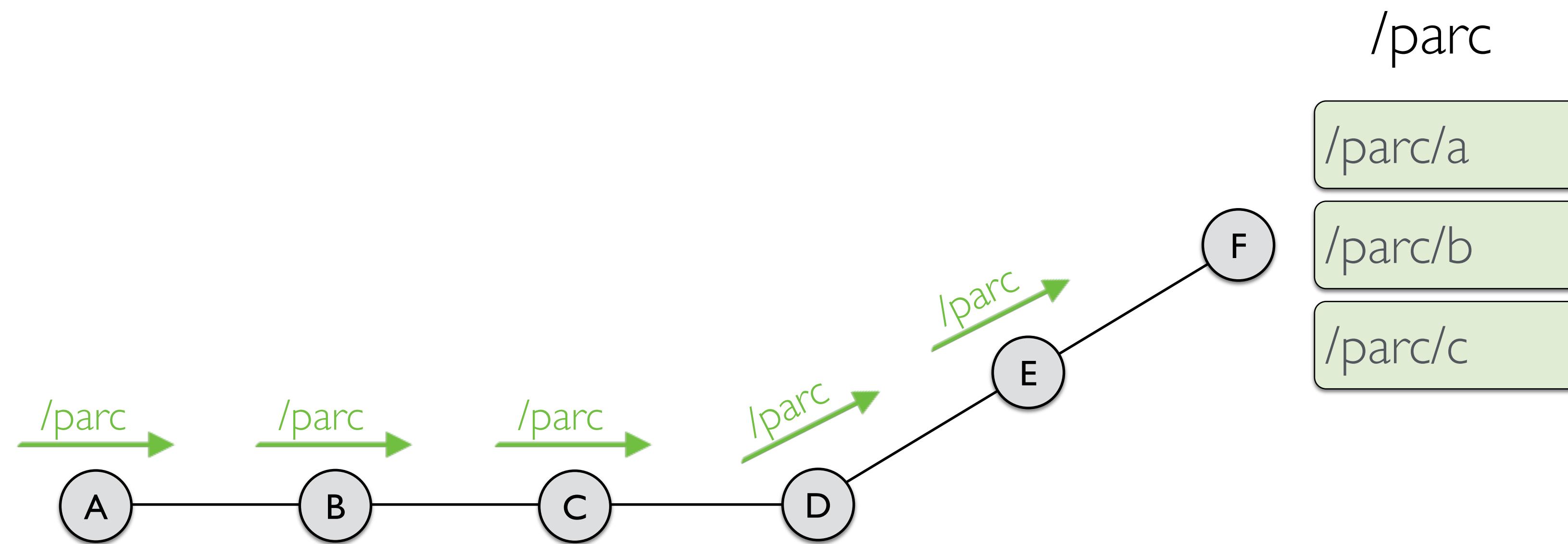
CCN - Base Protocol

Name routes are advertised



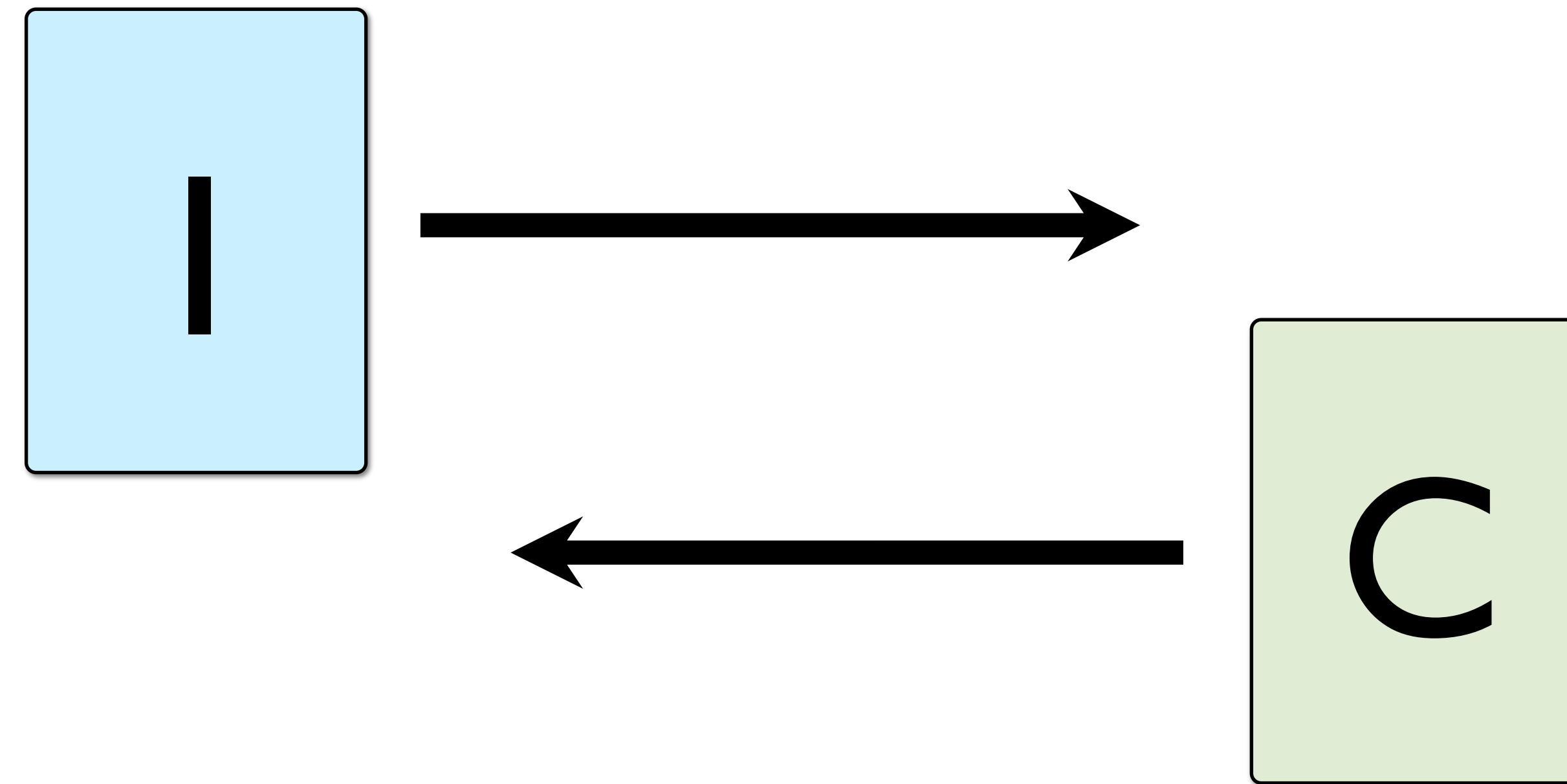
Node F serves /parc content.
Advertises a route to /parc

Name routes are set up in forwarders



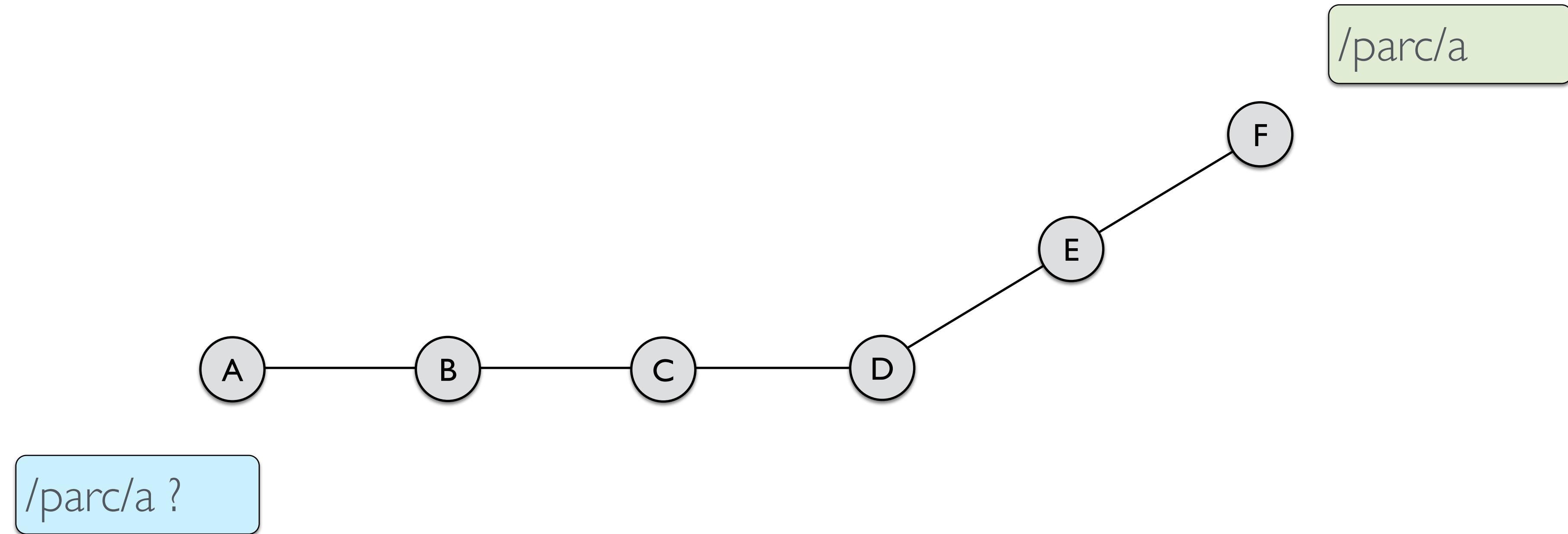
Routes are set up pointing to F for prefix /parc

Core Protocol



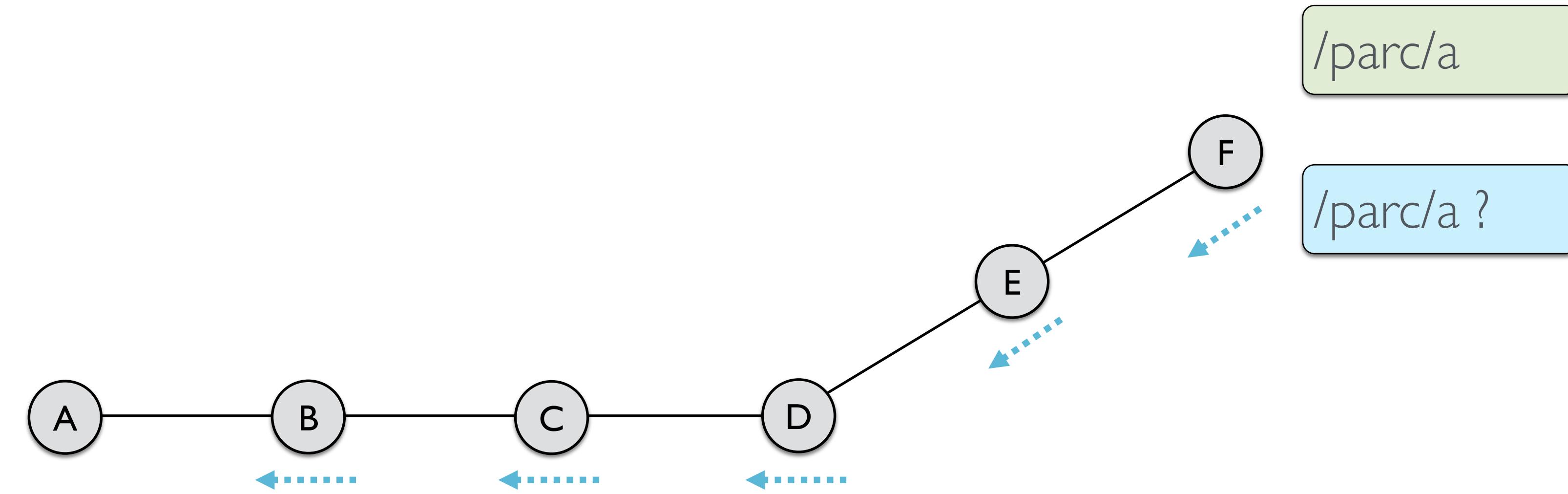
One interest packet gets one content packet

Interests follow the routes



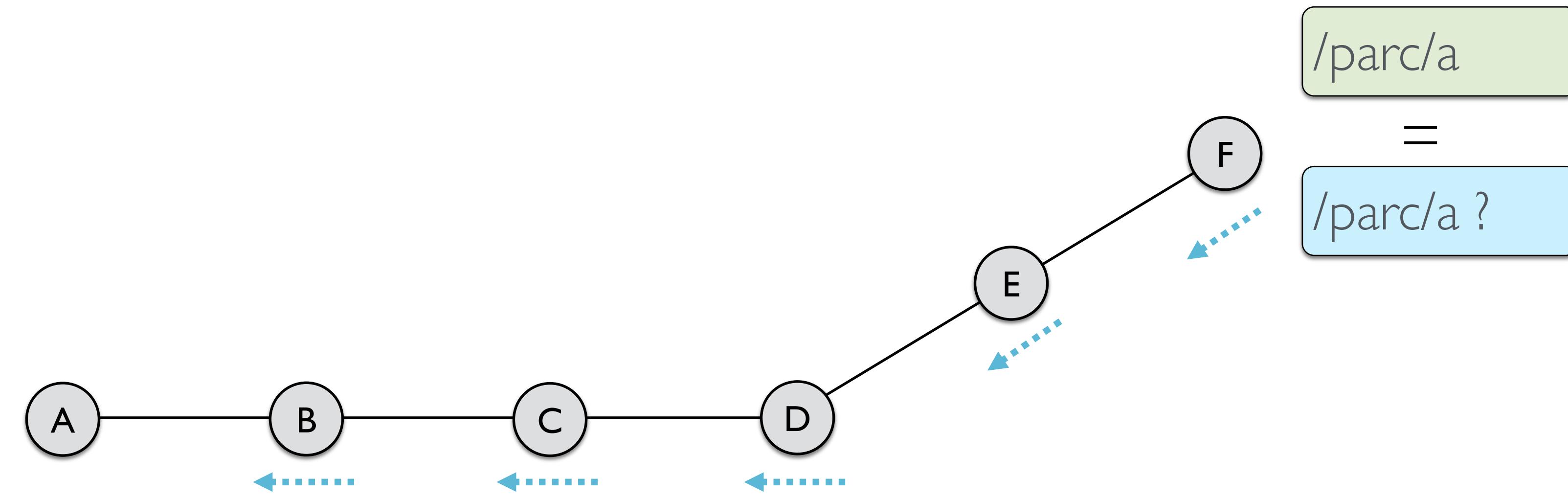
A issues interest for /parc/a

Interests leave breadcrumbs



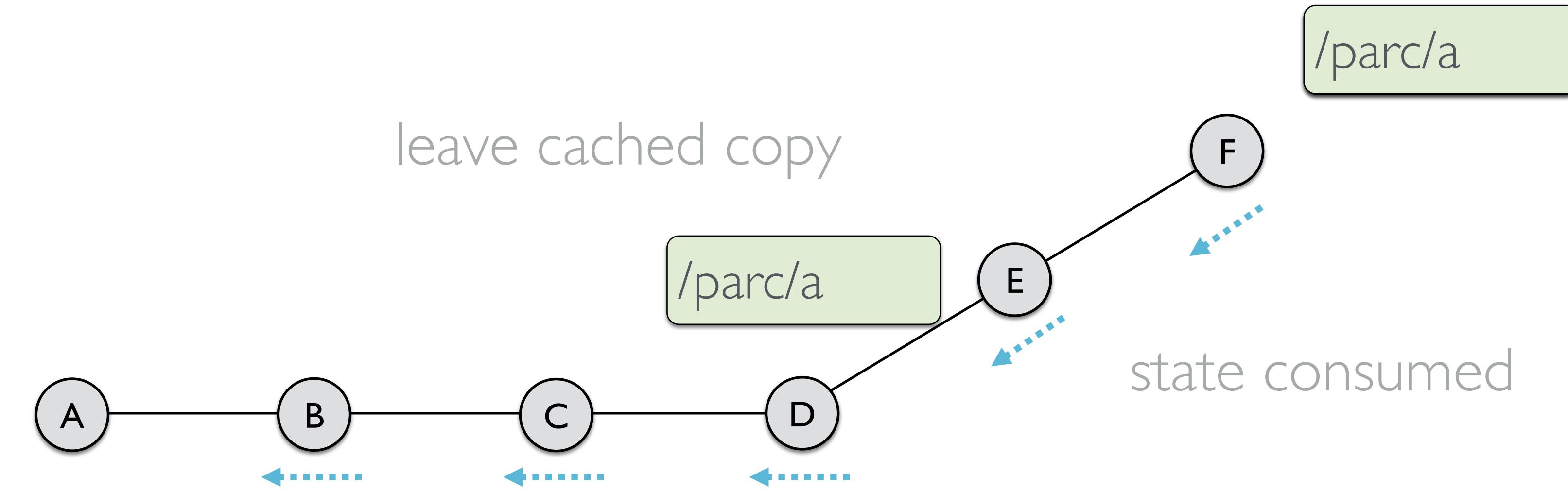
Interest leaves reverse path state in the network

Interest matches content



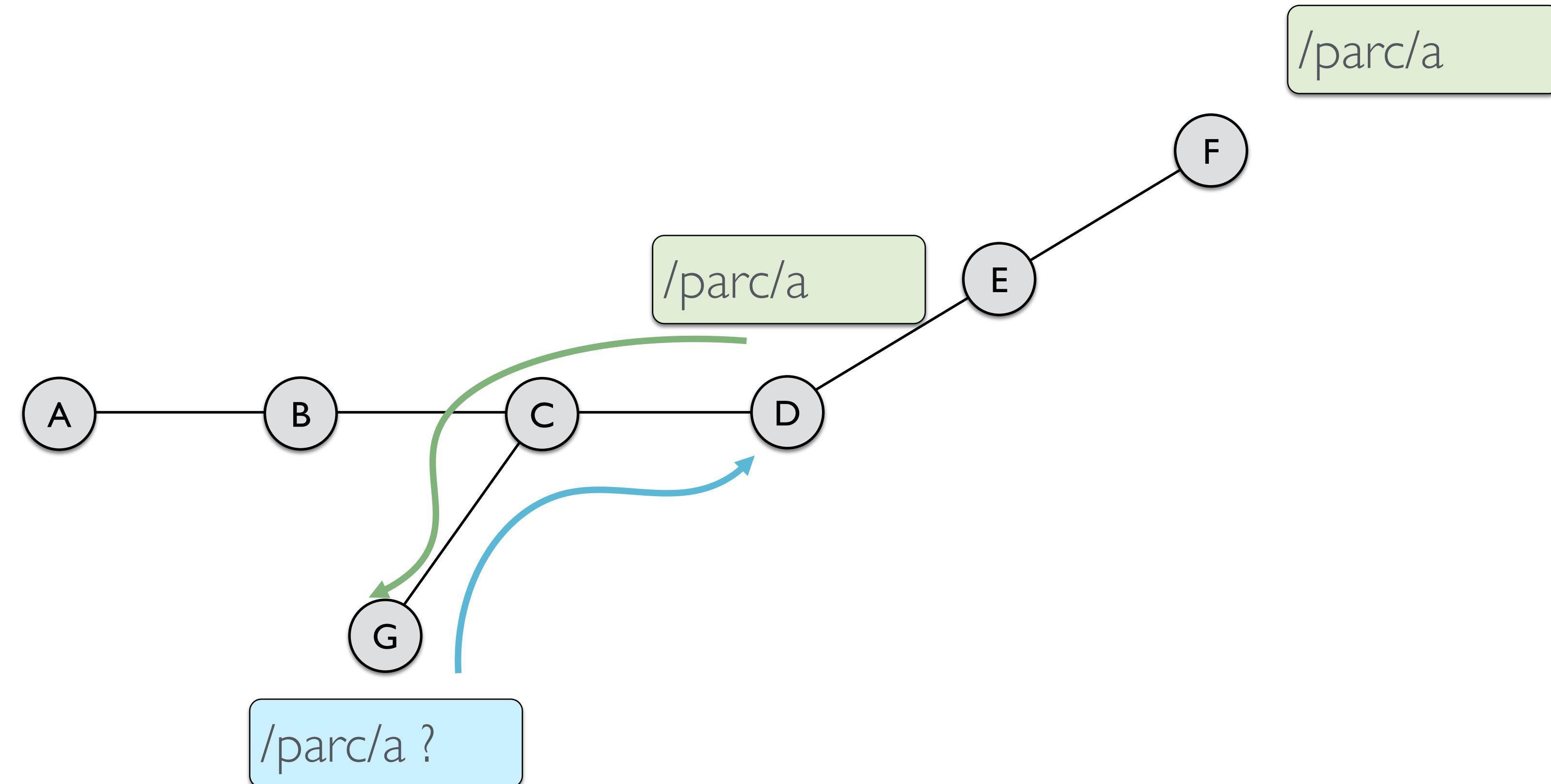
Interest for /parc/a finds a match at F

Content follows reverse path



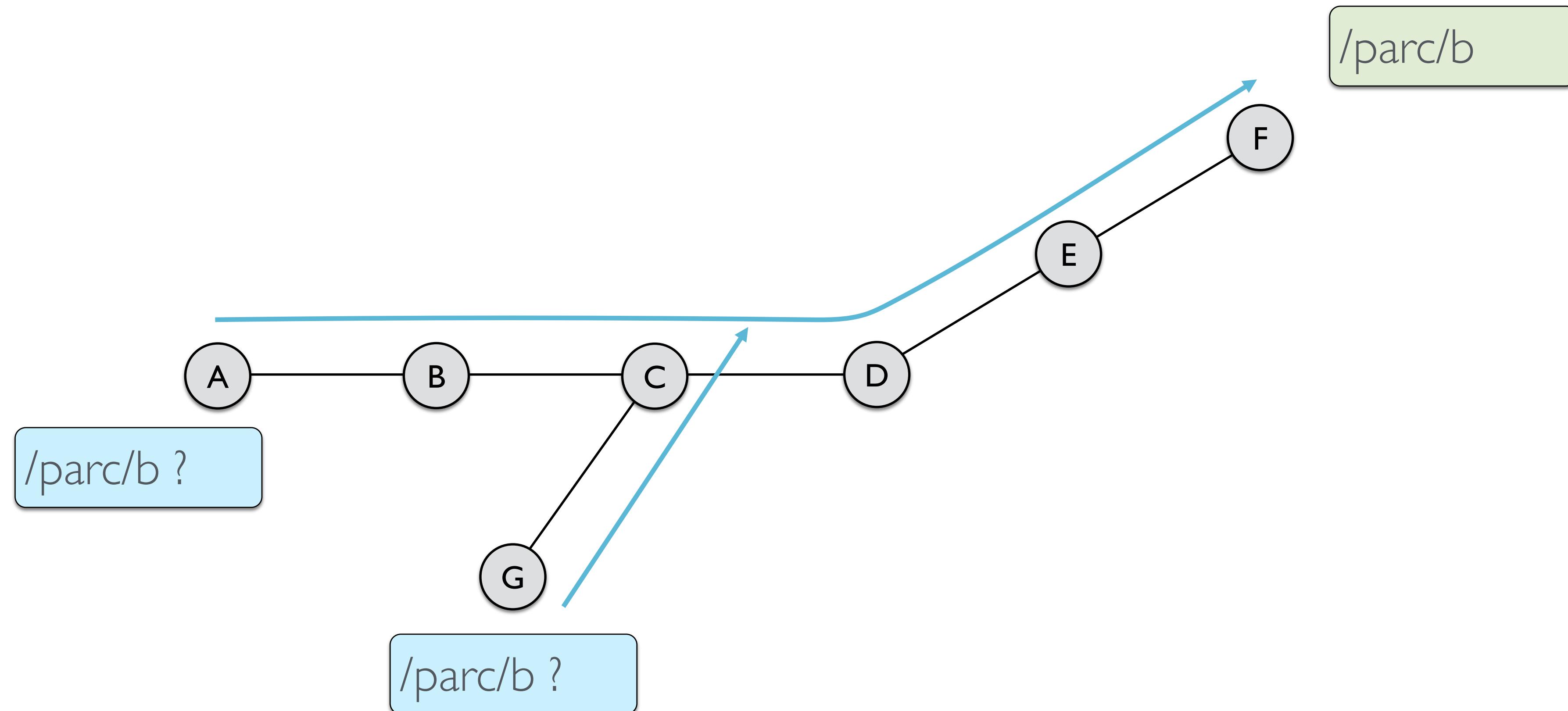
Content packet follows reverse path and consumes state

Any node can cache



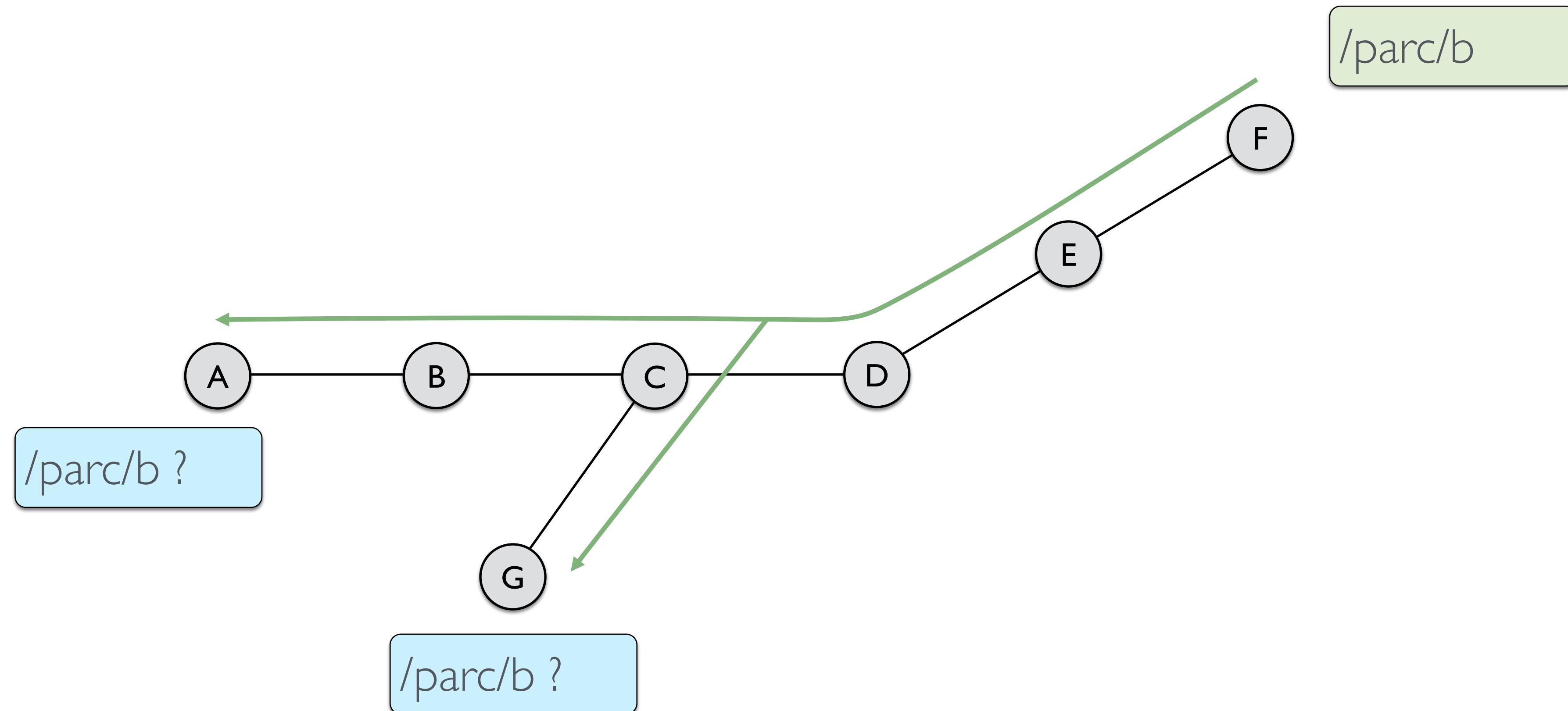
Node G requests same content, gets it from a cache in D
G authenticates content via the content signature, not the sender

Interests are aggregated



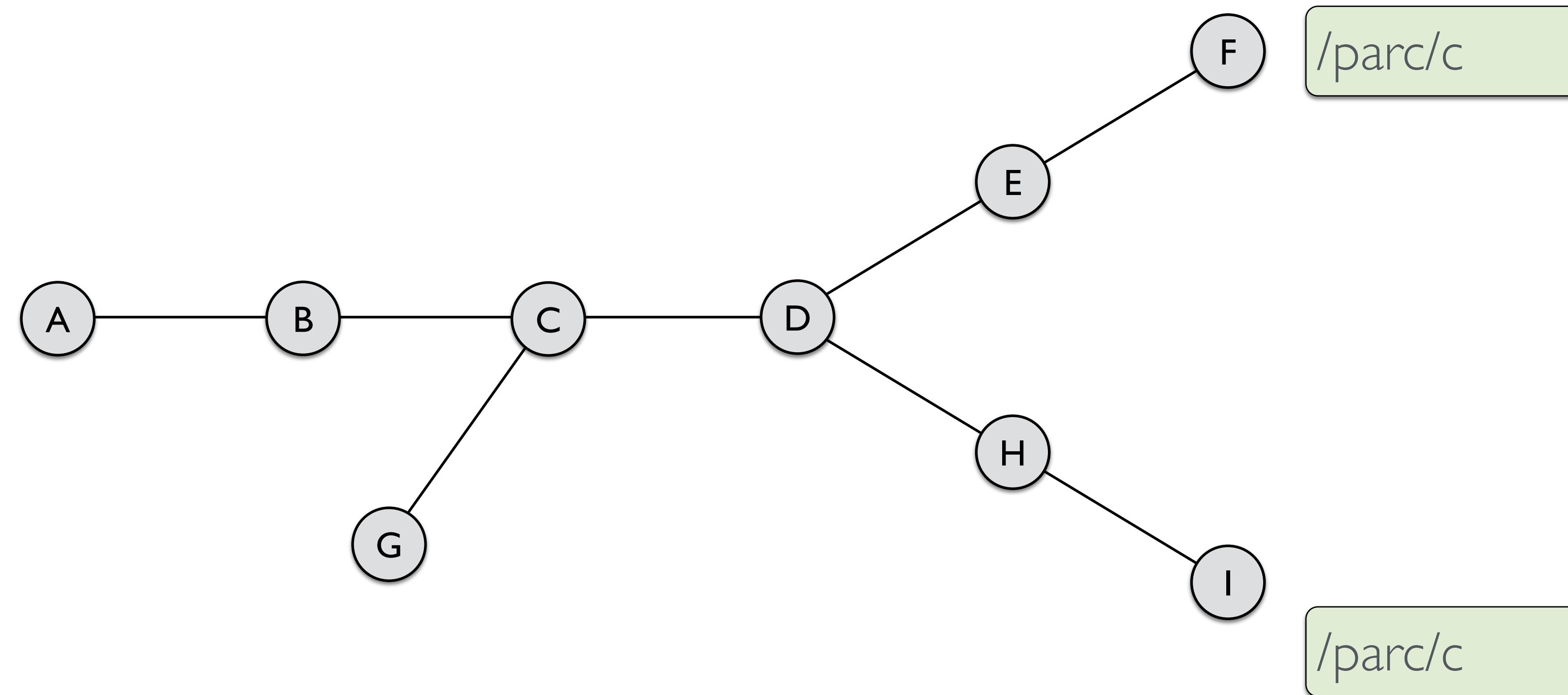
Node C aggregates the interests (only sends one upstream)

Content is sent in all interest paths



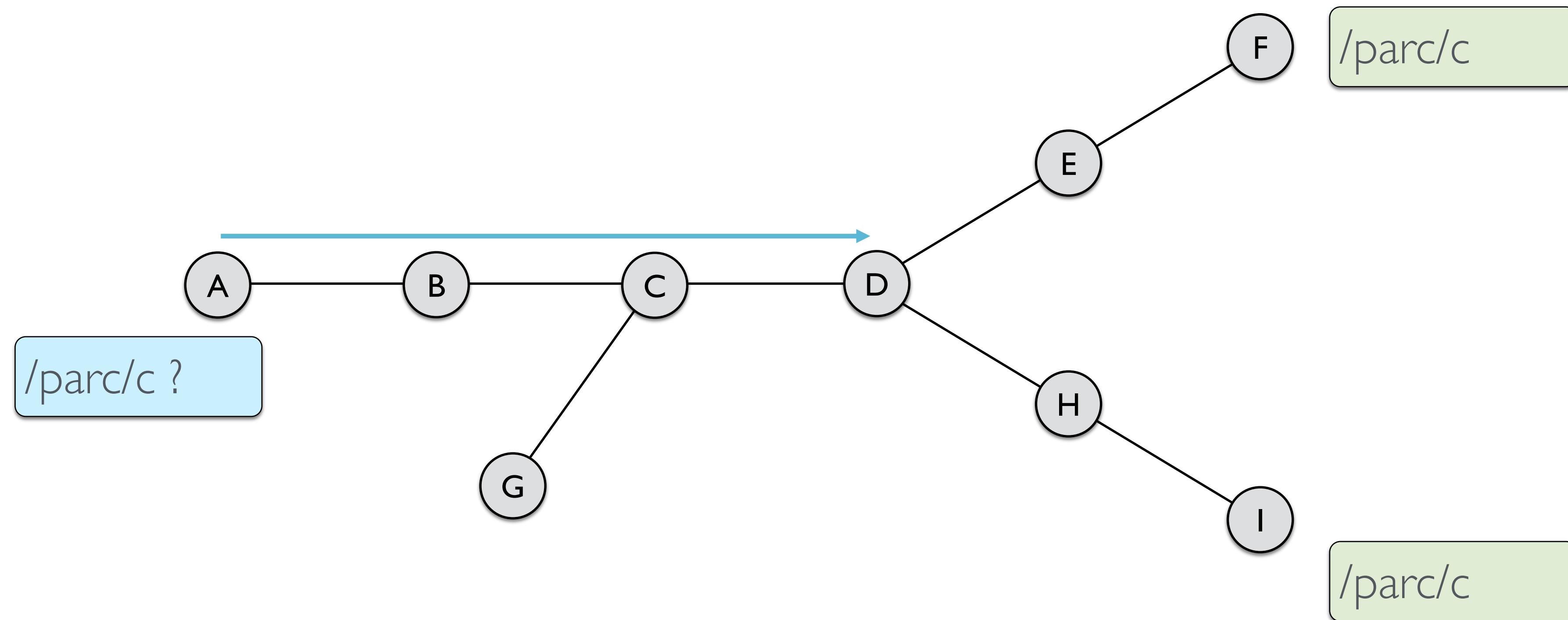
Node C sends the content packet to both B and G

Multiple nodes can advertise same prefix



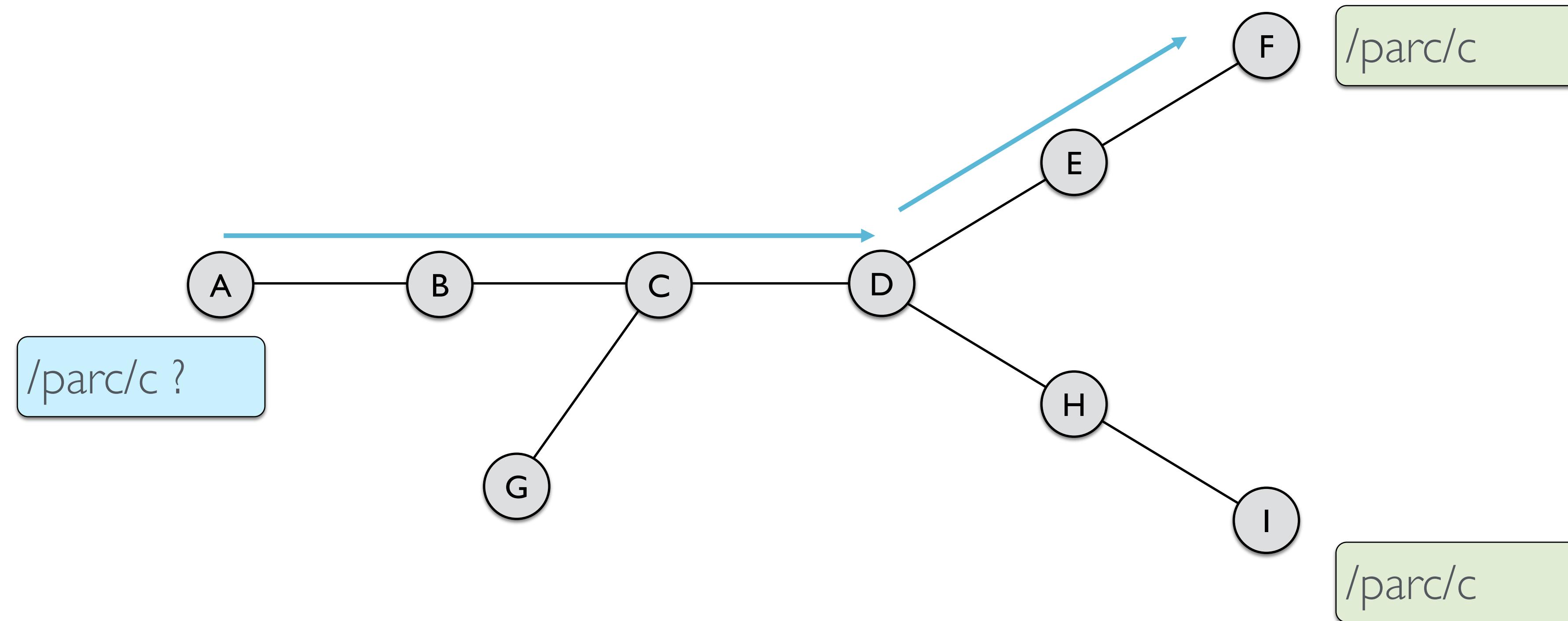
Both F and I can provide and advertise the content for /parc

Nodes can forward on either path



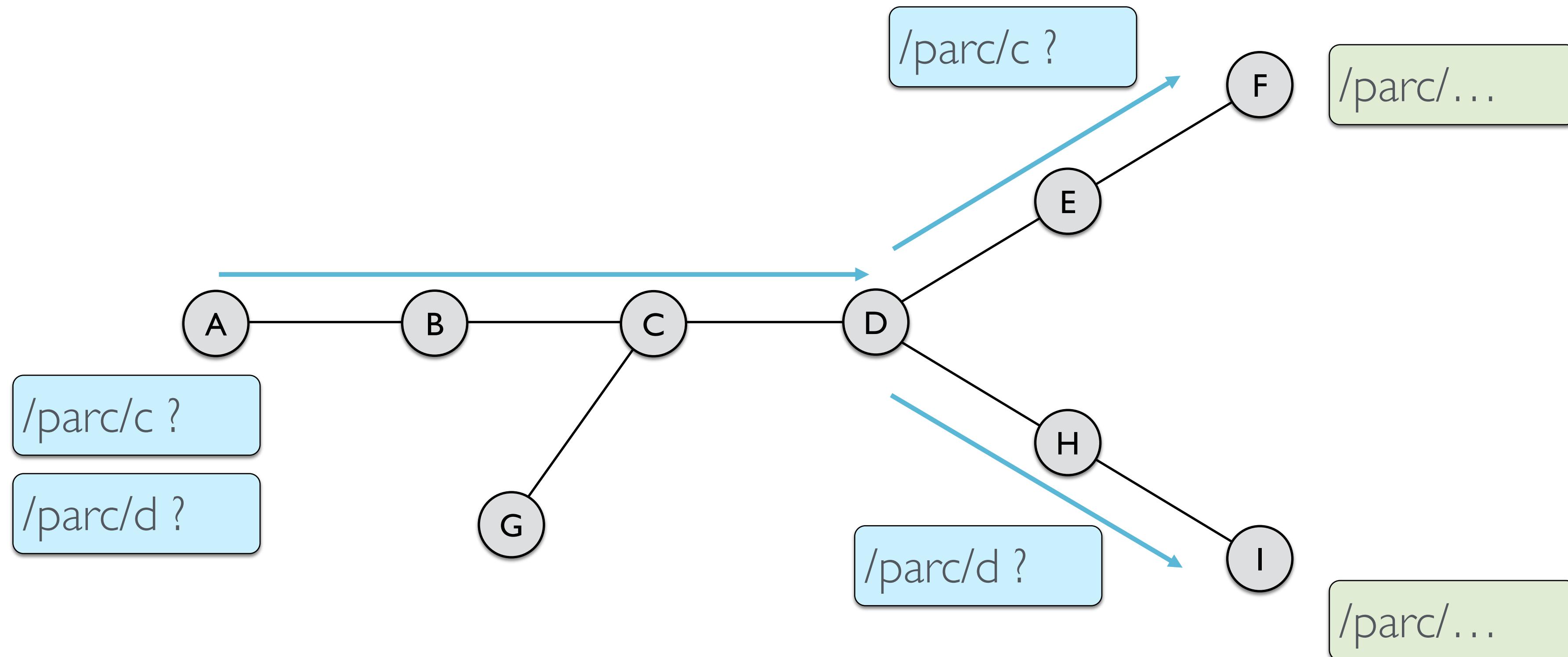
Nodes can choose where to send each interest

Forward on “best” path



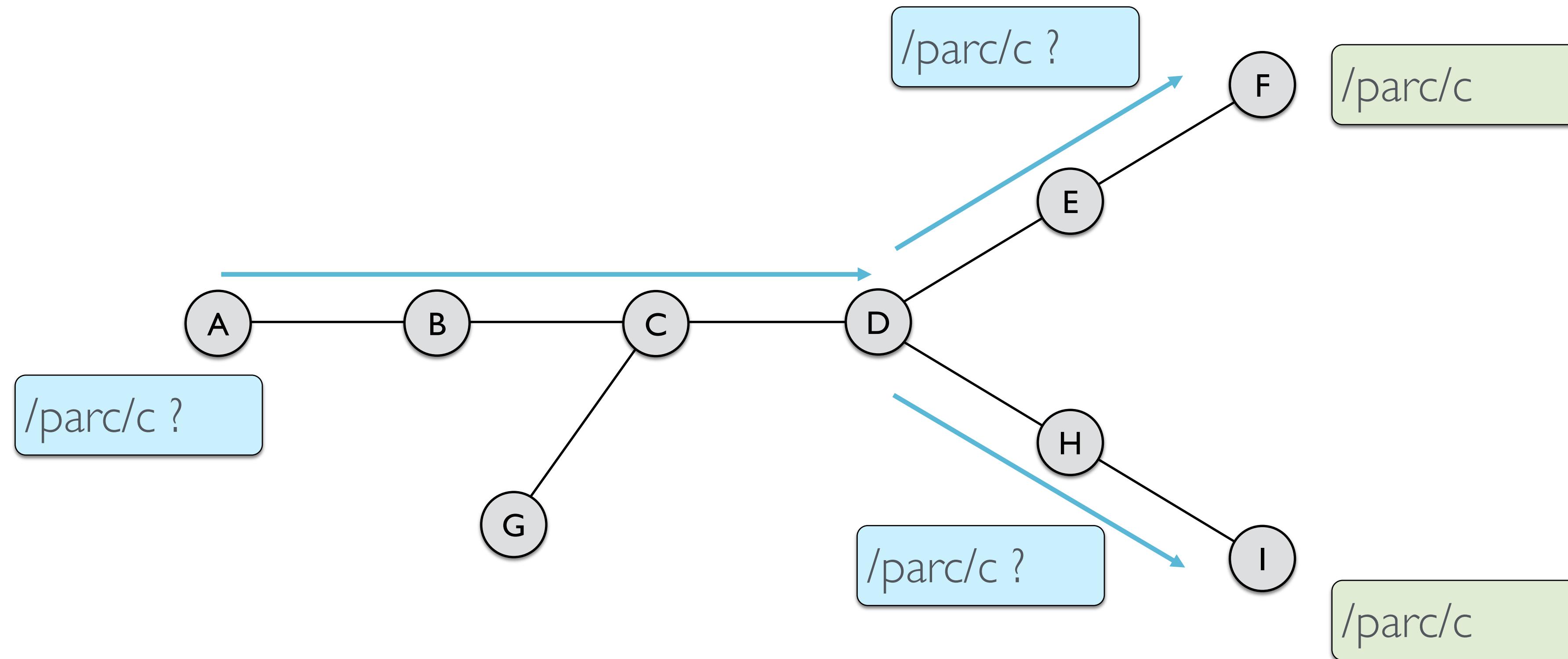
Choose one path to forward on

Forward on alternating paths



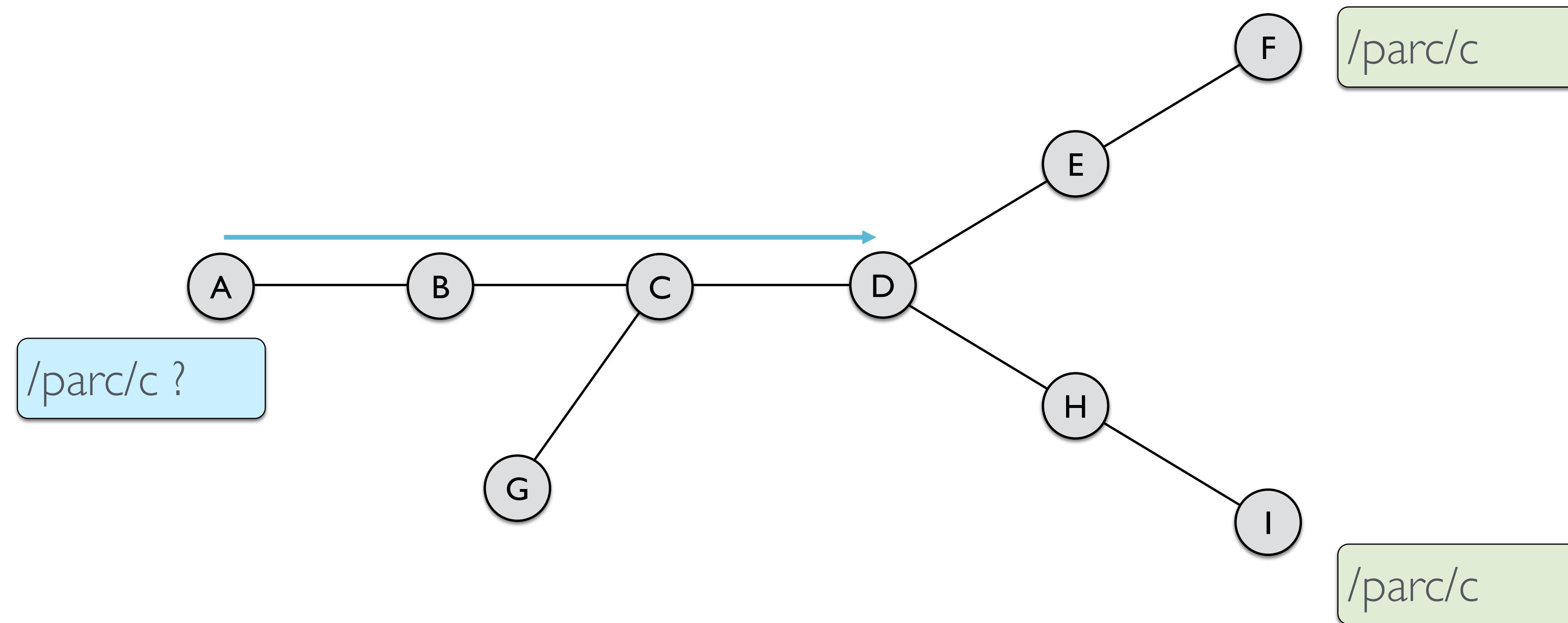
CCN is not connection oriented
Each interest can go to a different content provider

Forward on both paths



Node D will only reply to node C once
This can provide redundancy (at a high price)

Strategy Layer determines forwarding



The Strategy Layer at a forwarder determines what forwarding policy a node follows.

CCN - 1.x

Packet Format

Static Header

Information required on every packet.
May be modified at intermediate hops.

CCN I.x

Static Header

Optional Header

Optional information. May be modified
at intermediate hops.

Optional Headers

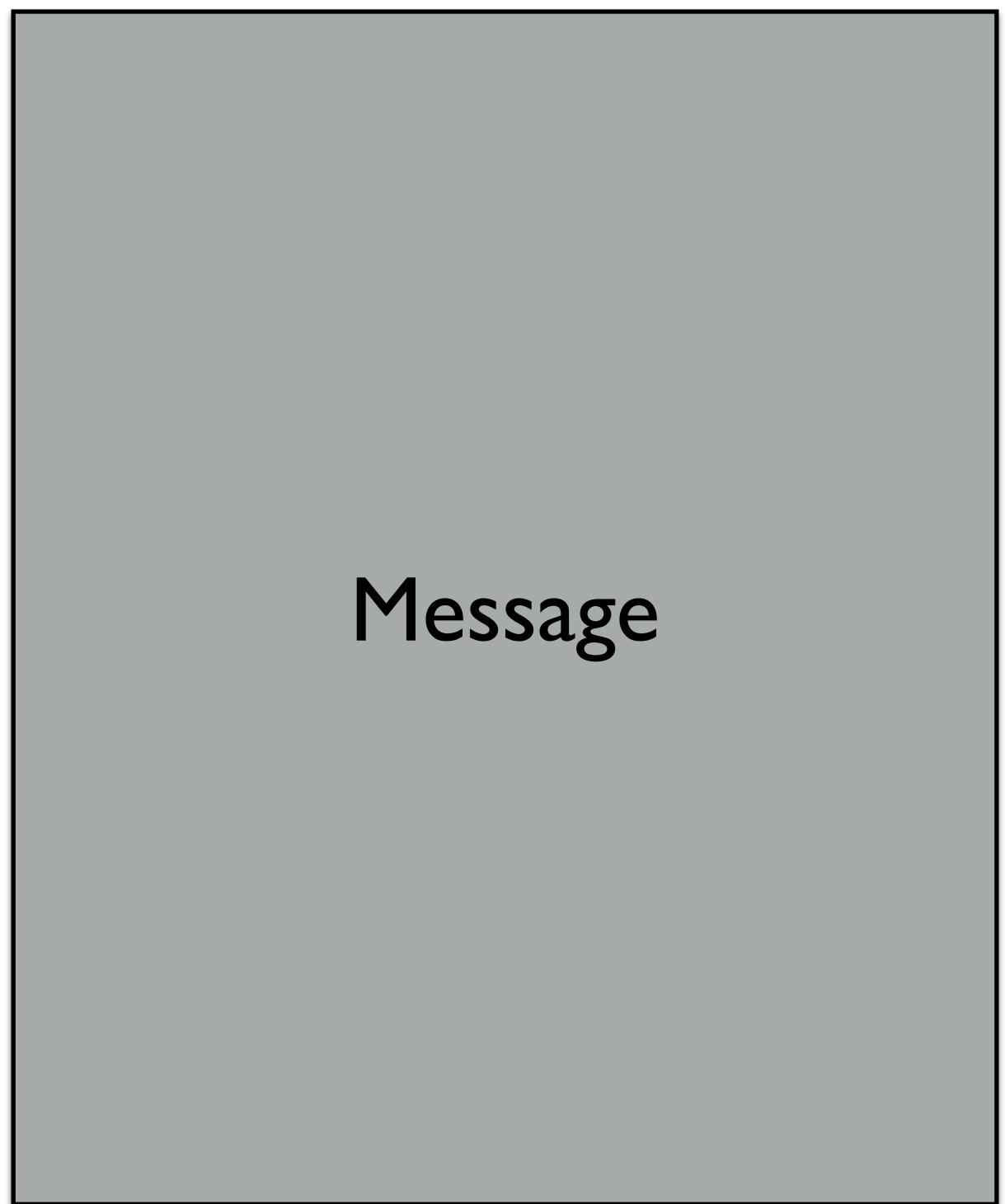
Message

Message

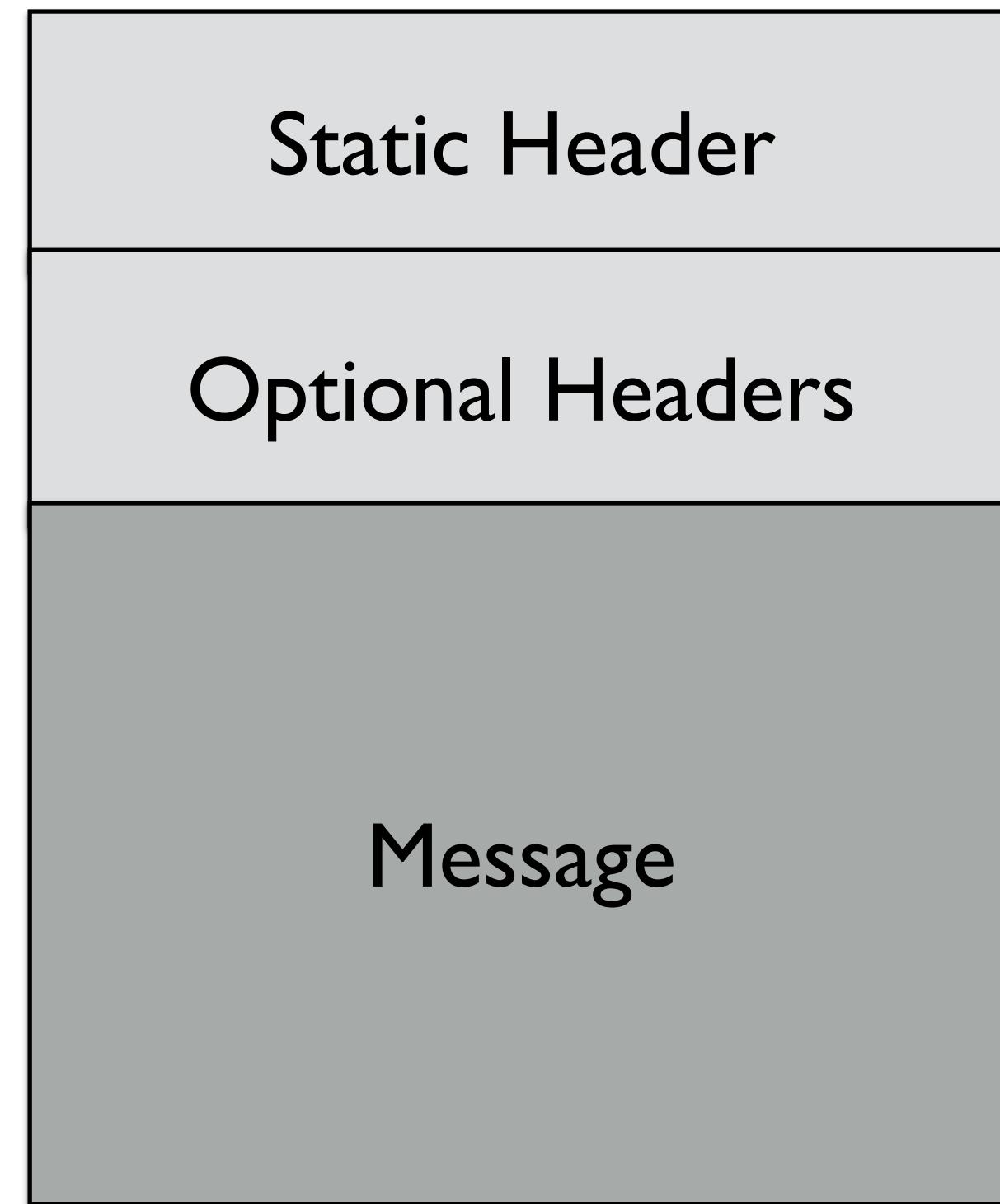
End-to-end protocol message.
Unmodified by intermediate elements.

Packet Format

CCN 0.x



CCN 1.x



Packet Format - Why change

Static header

- enables fast parsing
- contains common needs
- allows versioning

Optional headers

- allows network elements to add/modify information

Message Organization

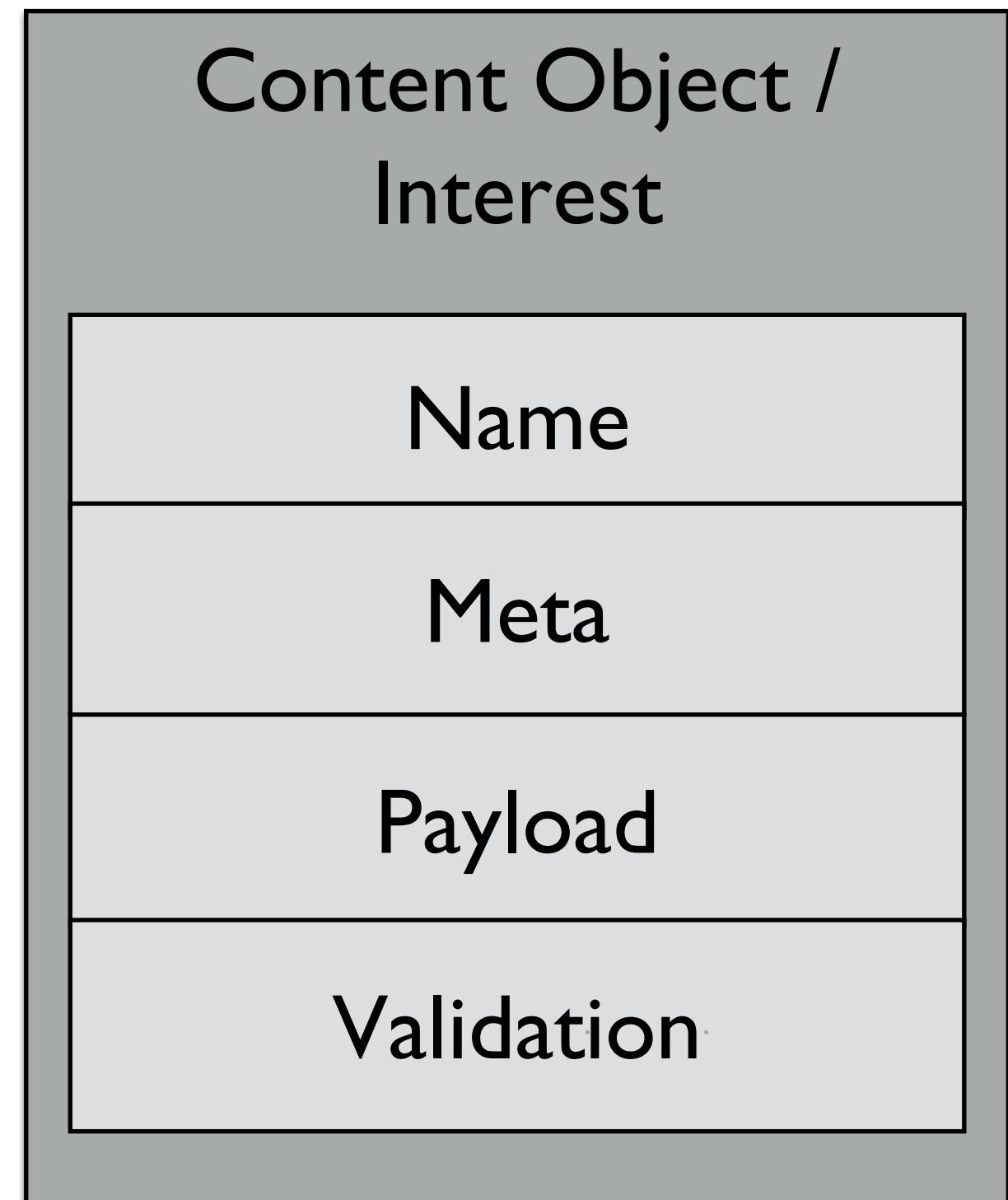
CCN I.x

Single message format

Name in front

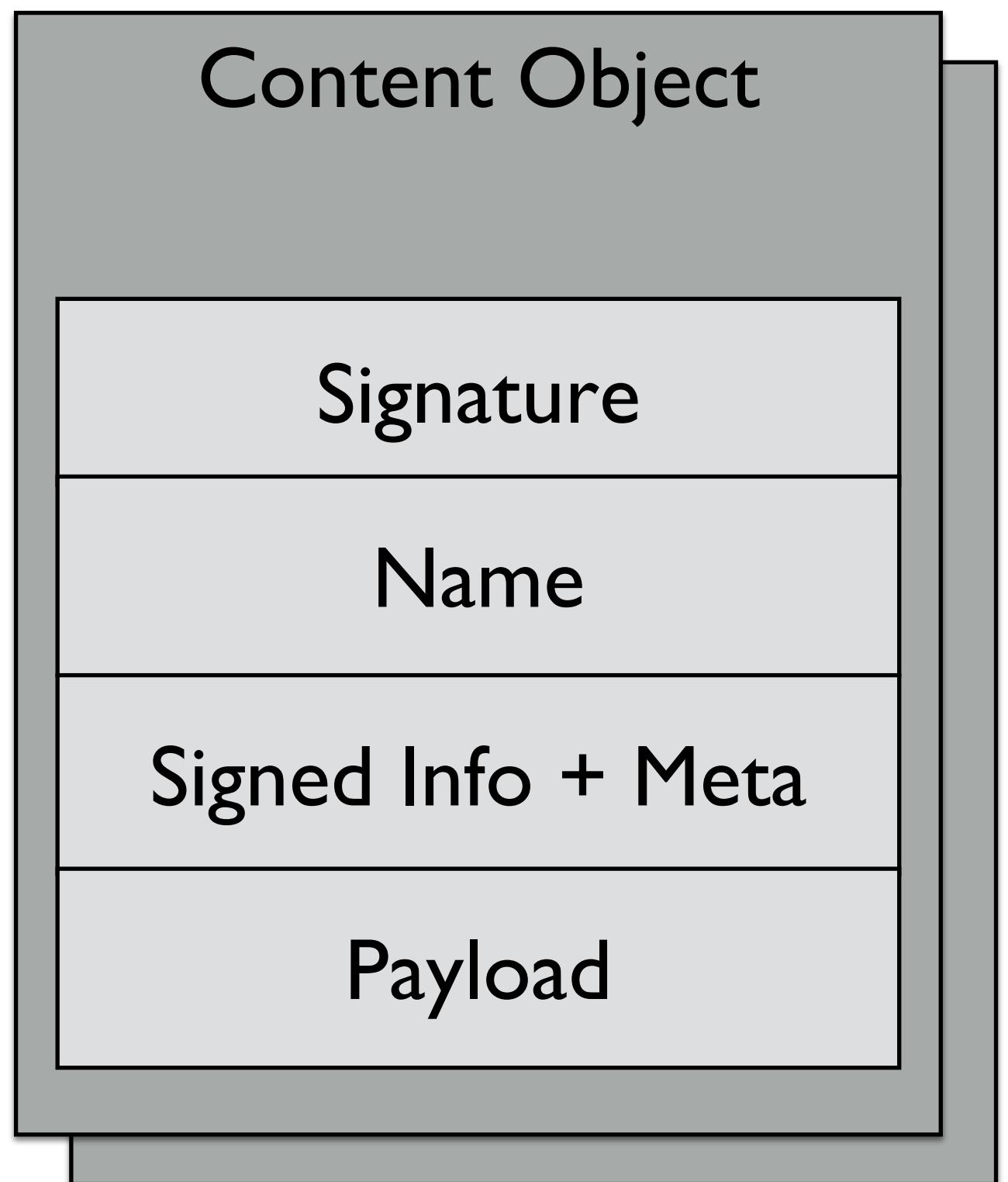
Payload

General Validation info in back

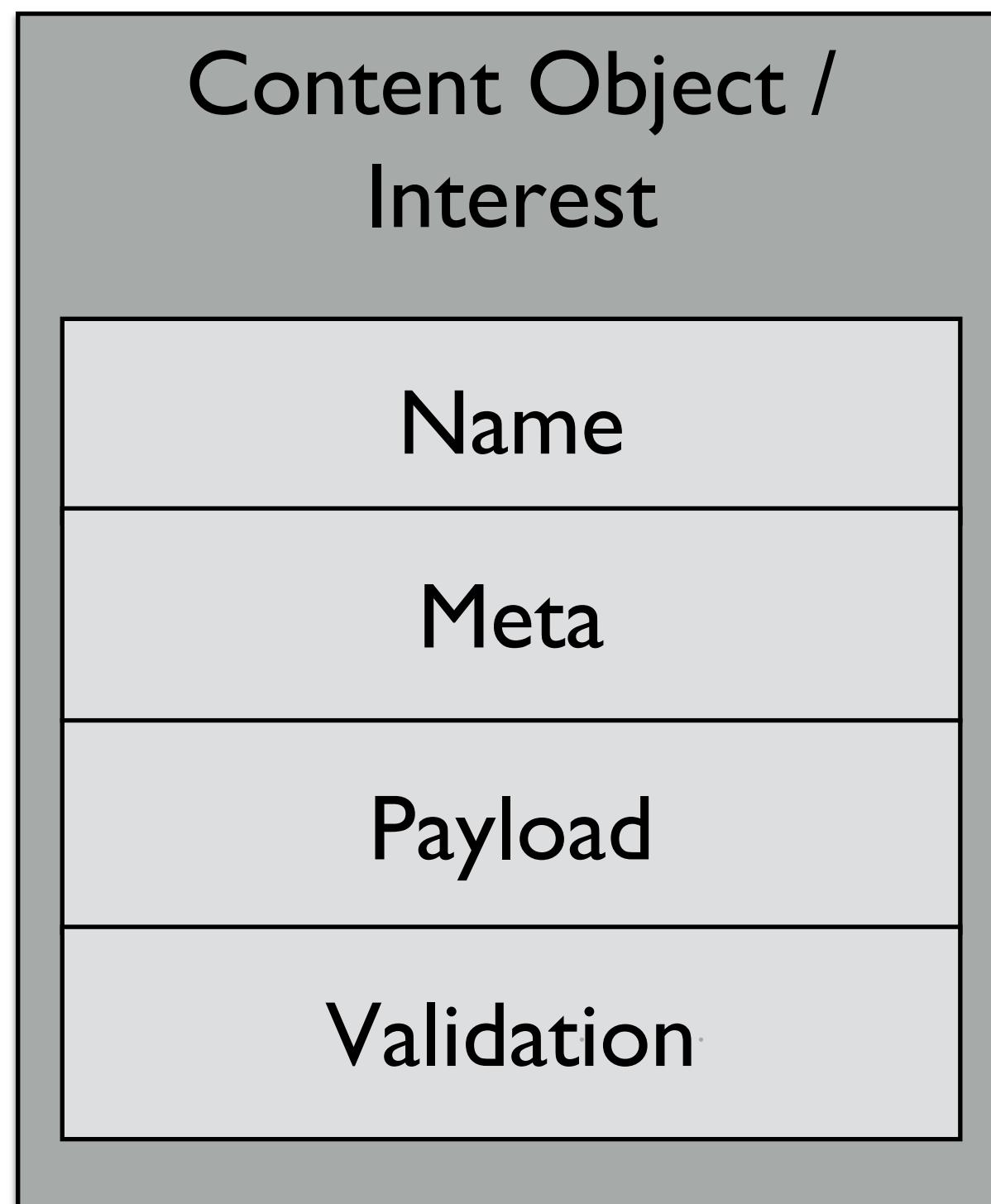


Message Organization

CCN 0.x



CCN 1.x



Message Organization - Why change

Name comes first

fast parsing

Separate validation from metadata at the end

modular security

Unified packet format

simplified, fast parsing

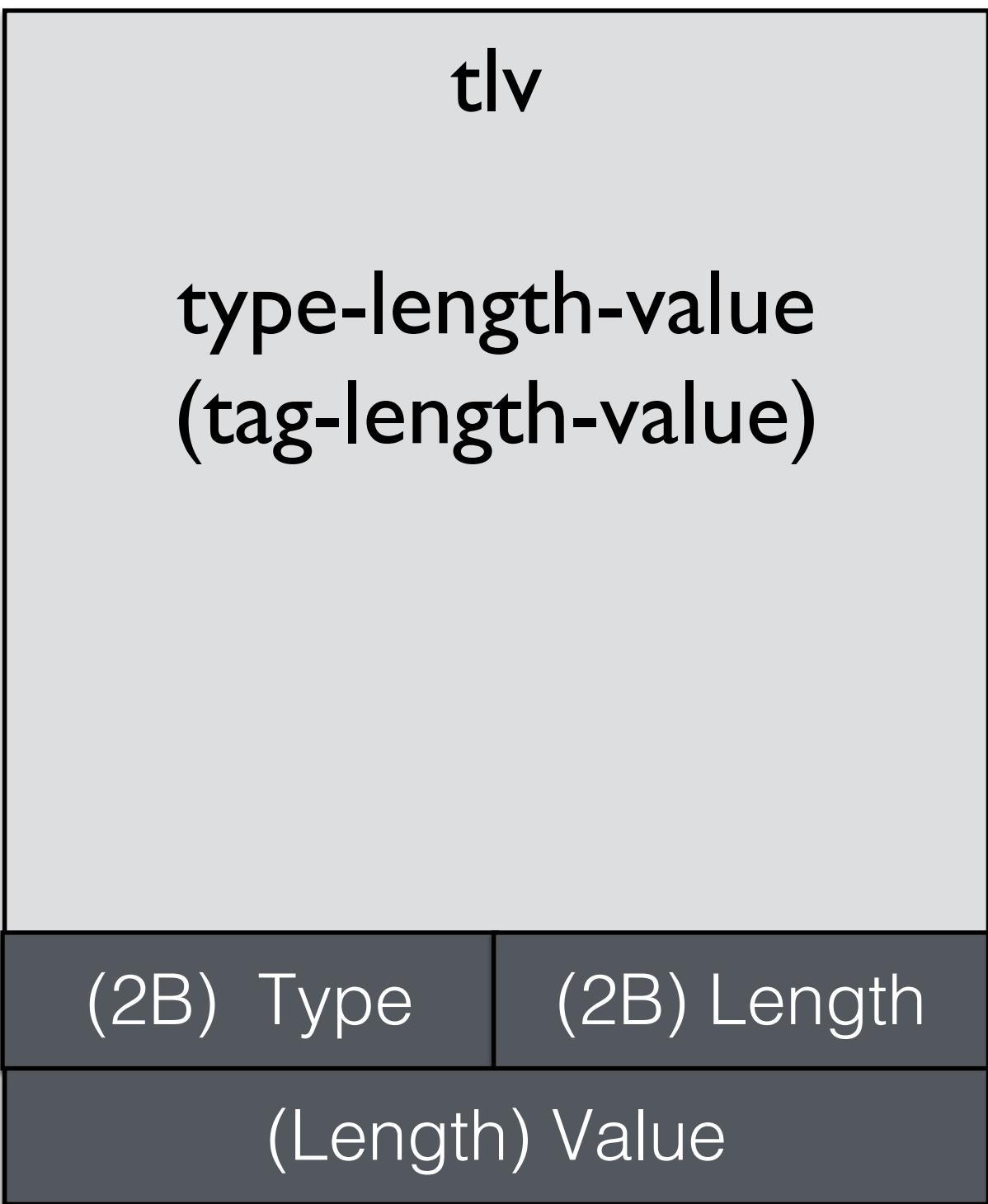
Packet Encoding

CCN I.x

Easy to parse

Easy to skip

Unambiguous



Packet Encoding

CCN 0.x

ccnb

“Custom binary encoding format for XML to meet specific needs of CCNx”

<block><block><block>

CCN 1.x

tlv

type-length-value
(tag-length-value)



Packet Encoding - Why change

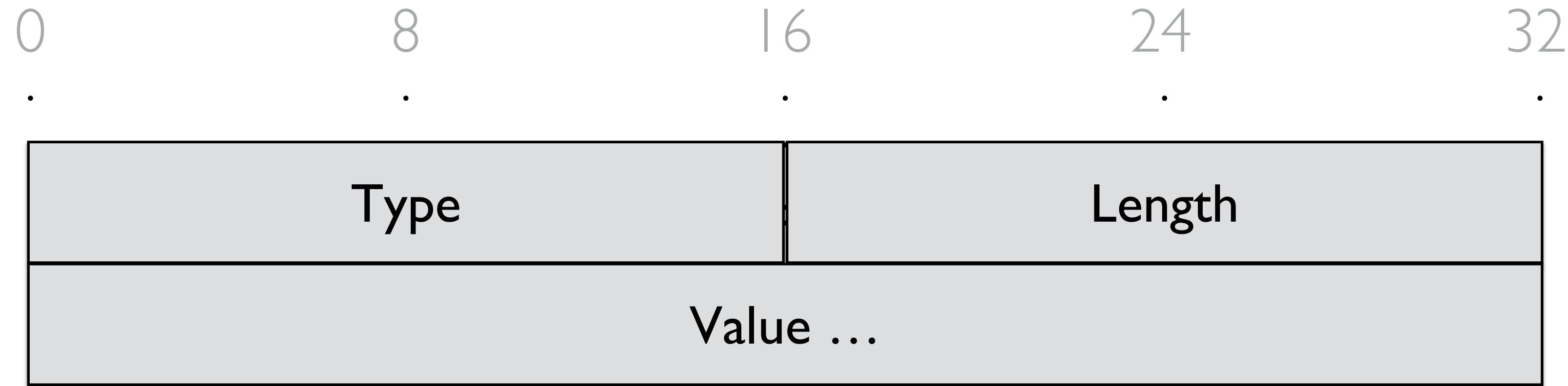
ccnb

flexible but complicated
relies on meta-structure
bit efficient

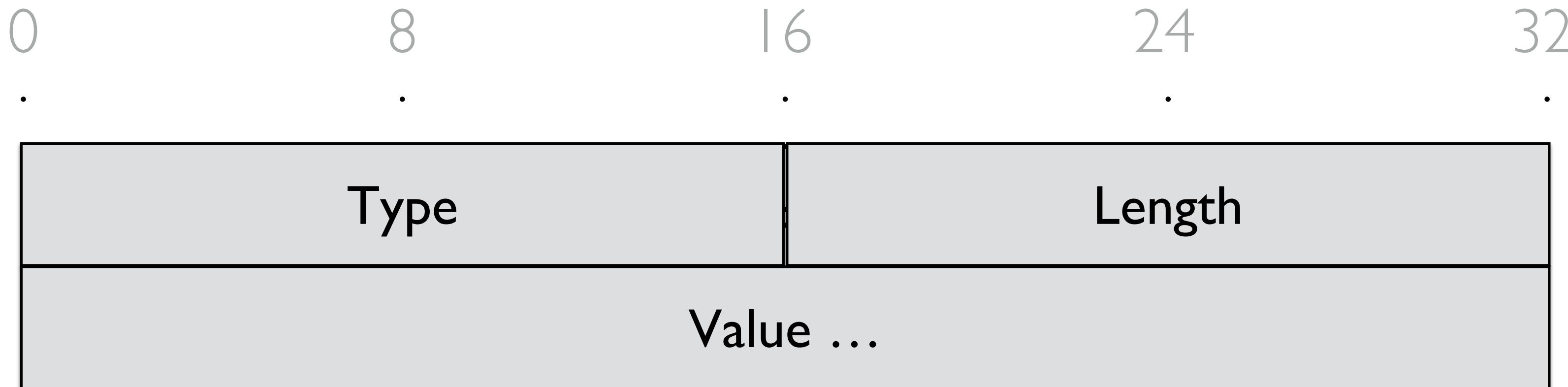
TLV

easy to parse
well understood
parse efficient

2 x 2 encoding



2 x 2 encoding



2-Byte Type

Enough for expansion and experiments
Inherently hierarchical (not many types)
Fast to jump over (no parsing needed)
No aliasing (0 vs 00)
Canonical sorting

2-Byte Length

Don't cover what doesn't fit in a packet
Enough for large fields like payload
Fast to jump over (no parsing)
No aliasing (0 vs 00)
Canonical sorting

Label-based names

CCN 1.x

All segments have types

/parc/ccn.zip/app<id>=1234/v=12/c=2/

Types specified by protocols

Type space for applications

Label-based names

CCN 0.x

/parc/ccn.zip/%C1.M.K%01%02/
%FD%04%62/%00%02/

CCN 1.x

/parc/ccn.zip/app<id>=1234/v=12/c=2/

Label-based names - Why change

No aliasing

Defining structure eliminates aliasing

Cleaner representation

More human readable

More powerful

Structure allows network elements to make choices

Matching (exact)

CCN I.x

/parc/ccn.zip == /parc/ccn.zip

Exact binary match

Matching (exact)

CCN 0.x

CCN 1.x

/parc/ccn.zip ==	/parc/ccn.zip	== /parc/ccn.zip
/parc/ccn.zip/v2/s3 ==		
/parc/ccn.zip/v2/s4 ==		
/parc/ccn.zip/v7/s6 ==		
/parc/ccn.zip/v100/s1 ==		
/parc/ccn.zip/meta/v1/s8 ==		
/parc/ccn.zip/discussion ==		
cn.zip/.acl/group/owner/key/v1/s9 ==		

Matching (no selectors)

CCN 0.x

Interest:

name = /parc/ccn.zip
minSuffixComponents=x
maxSuffixComponents=y
exclude=xxx,xxx,xxx,...
childSelector=Left/Right

CCN 1.x

Interest:

name = /parc/ccn.zip

Matching - Why change

Exact match is deterministic

You get what you ask for

Efficient match

Fast forwarding on single match

No rummaging of caches / traffic

Better privacy

Matching (restrictions)

CCN 0.x

Interest:
name = /parc/ccn.zip
pubKeyDigest=xxx

Interest:
name = /parc/ccn.zip/abcd
minSuffixComponents=0
maxSuffixComponents=0

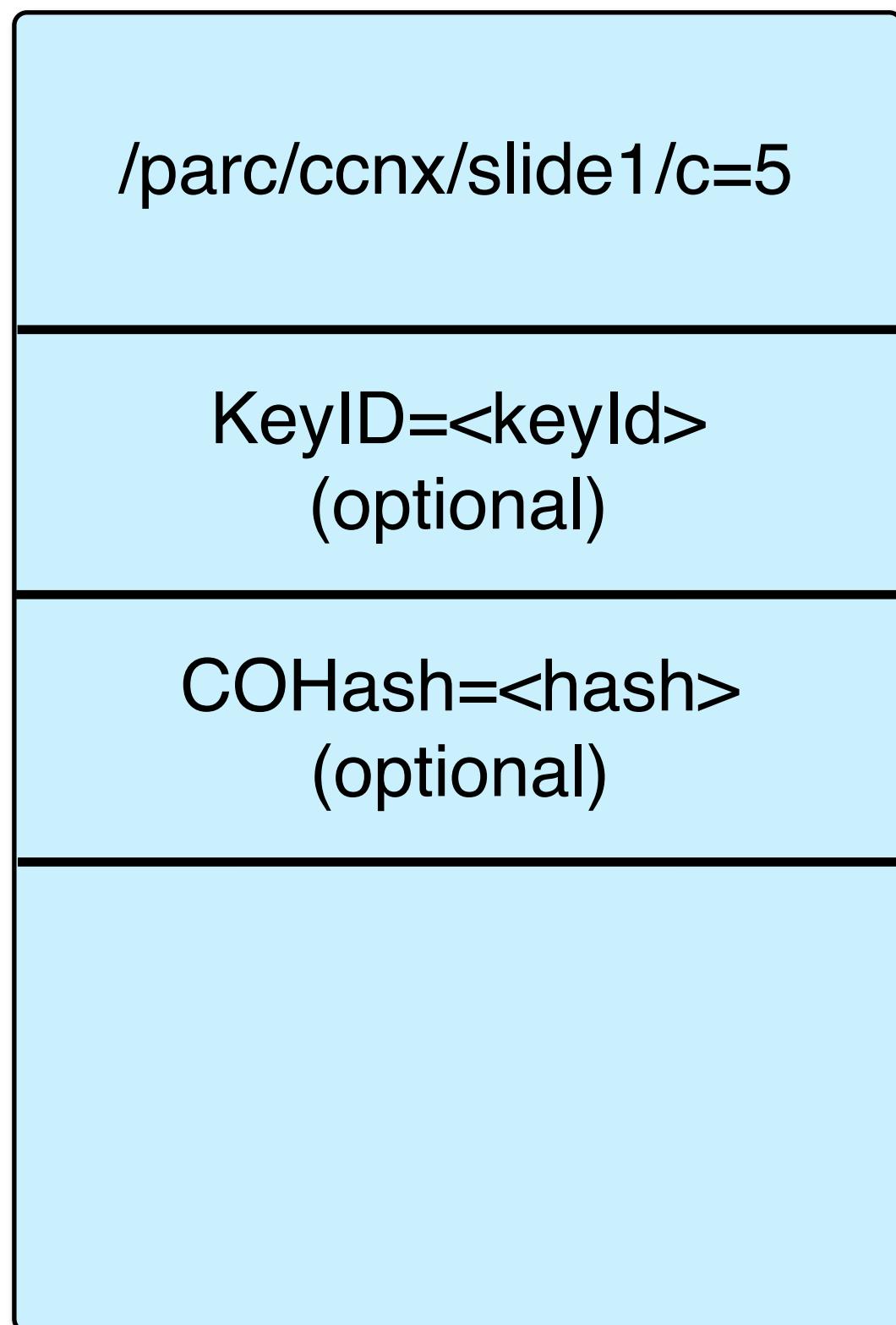
CCN 1.x

Interest:
name = /parc/ccn.zip
keyIdRestriction=xxx

Interest:
name = /parc/ccn.zip
contentObjectHash=abcd

Core Protocol Primitives

Interest

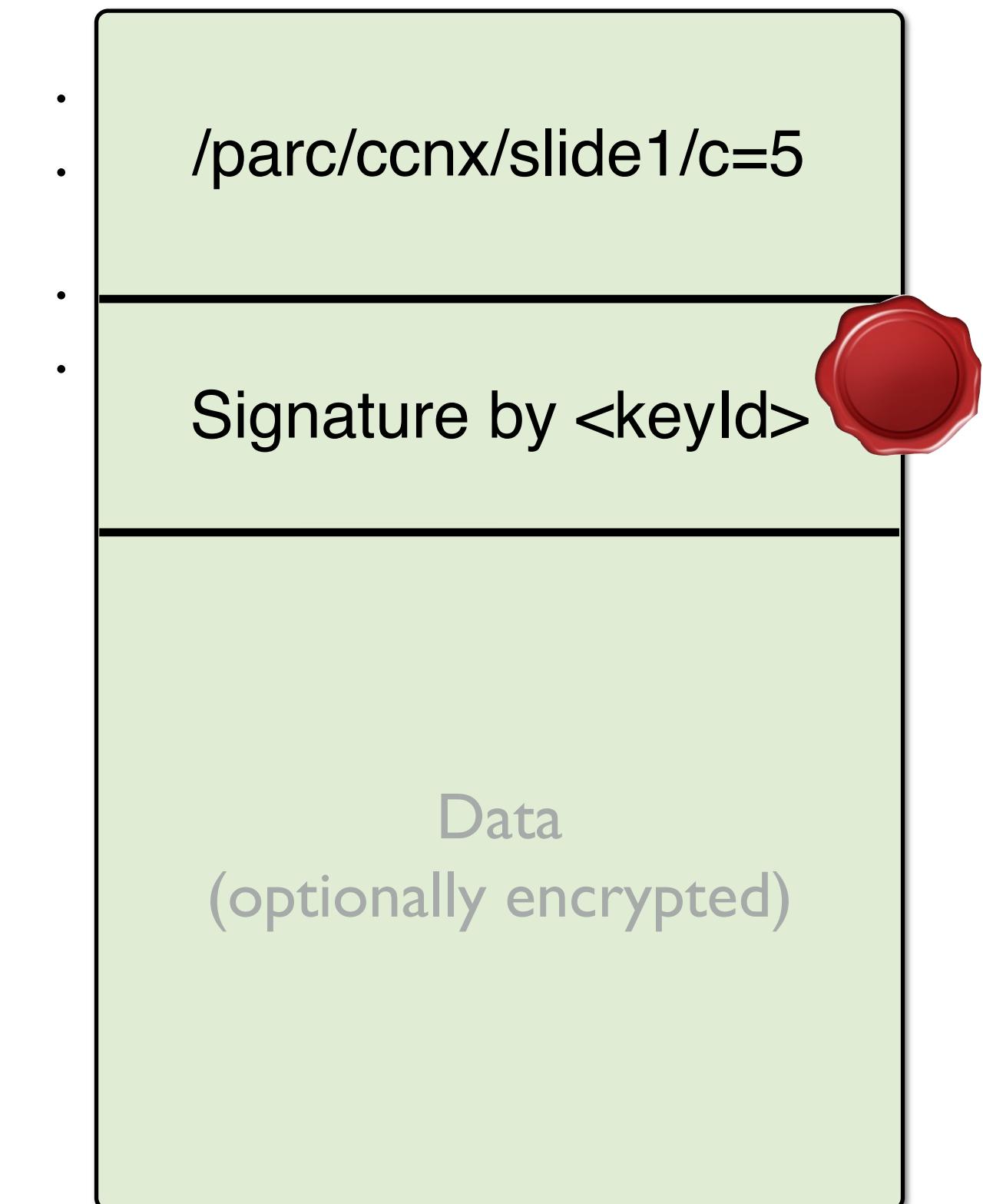


==

==

==

Content Object



hash

Matching (restrictions) - Why change

No ‘real’ change

Functionally the same

Explicit contentObjectHash matching

Simpler matching (not intermingled)

Loop halting

CCN 1.x

Loops halted by PIT

Hop-Limit is stop-gap

Interest:
name = /parc/ccn.zip
hop-limit = 16

PIT
/parc/ccn.zip

Loop halting

CCN 0.x

Interest:

name = /parc/ccn.zip
nonce = 1234

PIT

/parc/ccn.zip : 1234

CCN 1.x

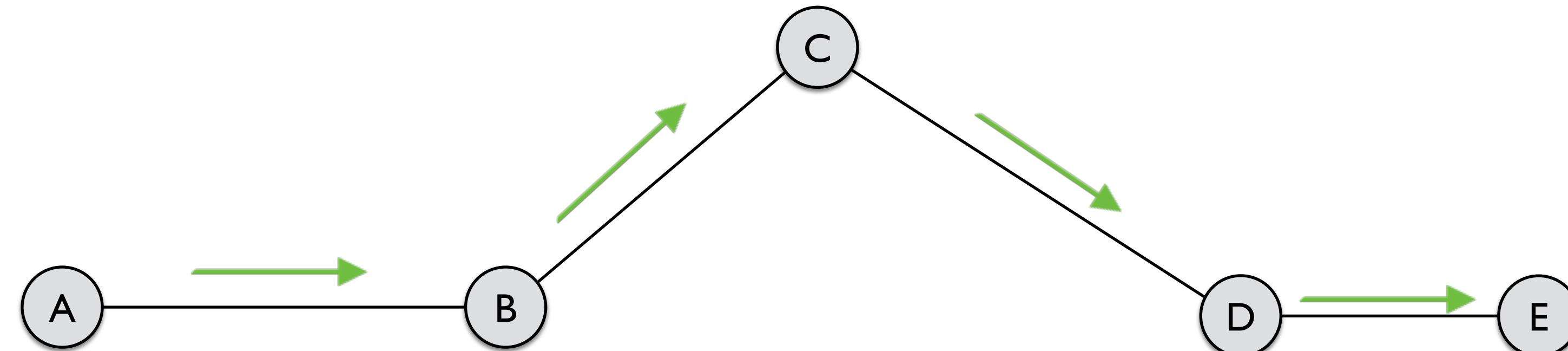
Interest:

name = /parc/ccn.zip
hop-limit = 16

PIT

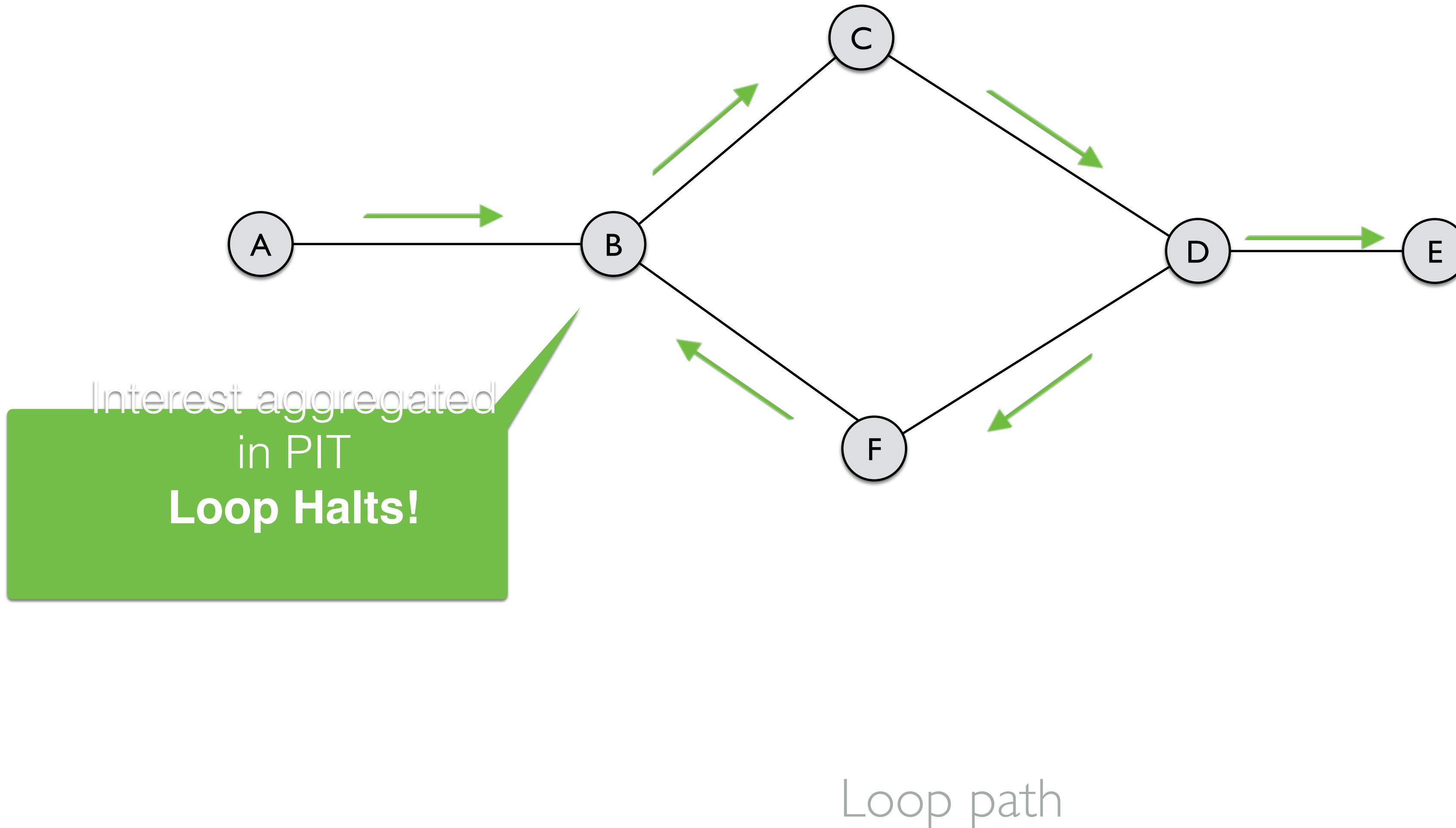
/parc/ccn.zip

Nonces

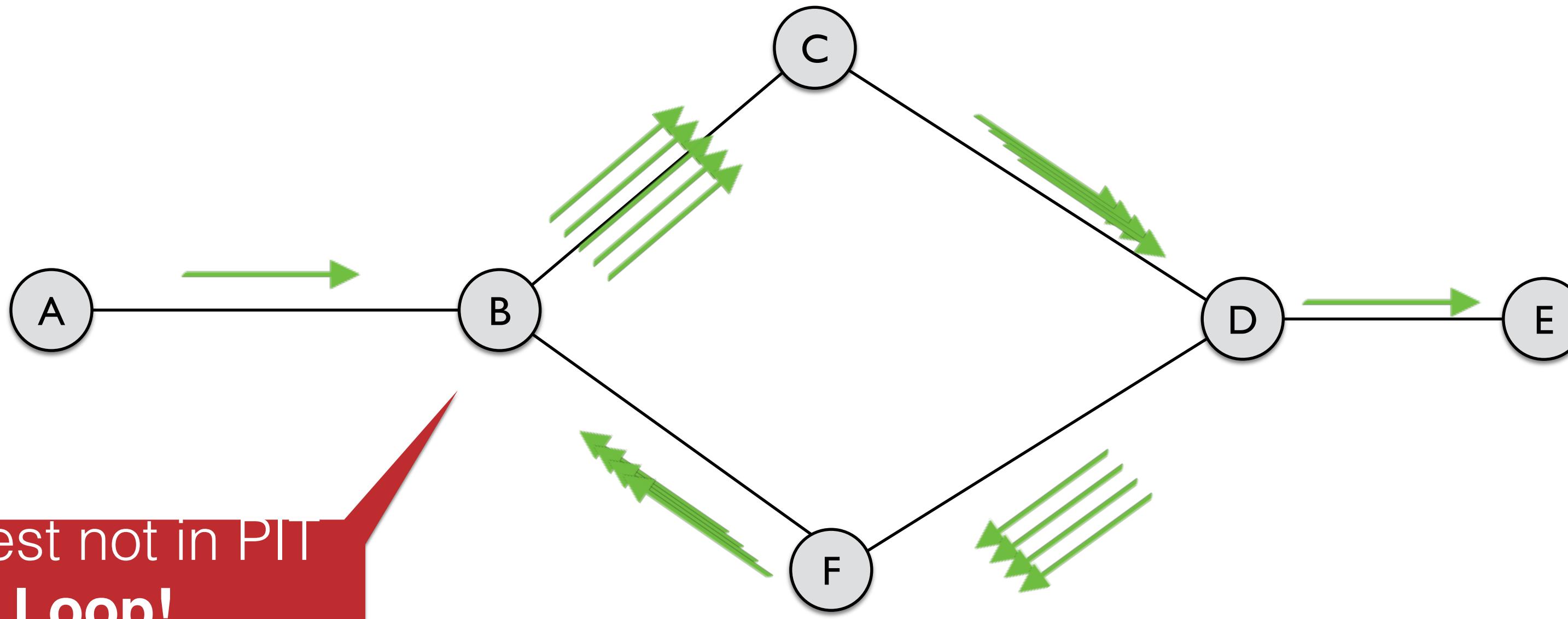


Regular path

Nonces and loops

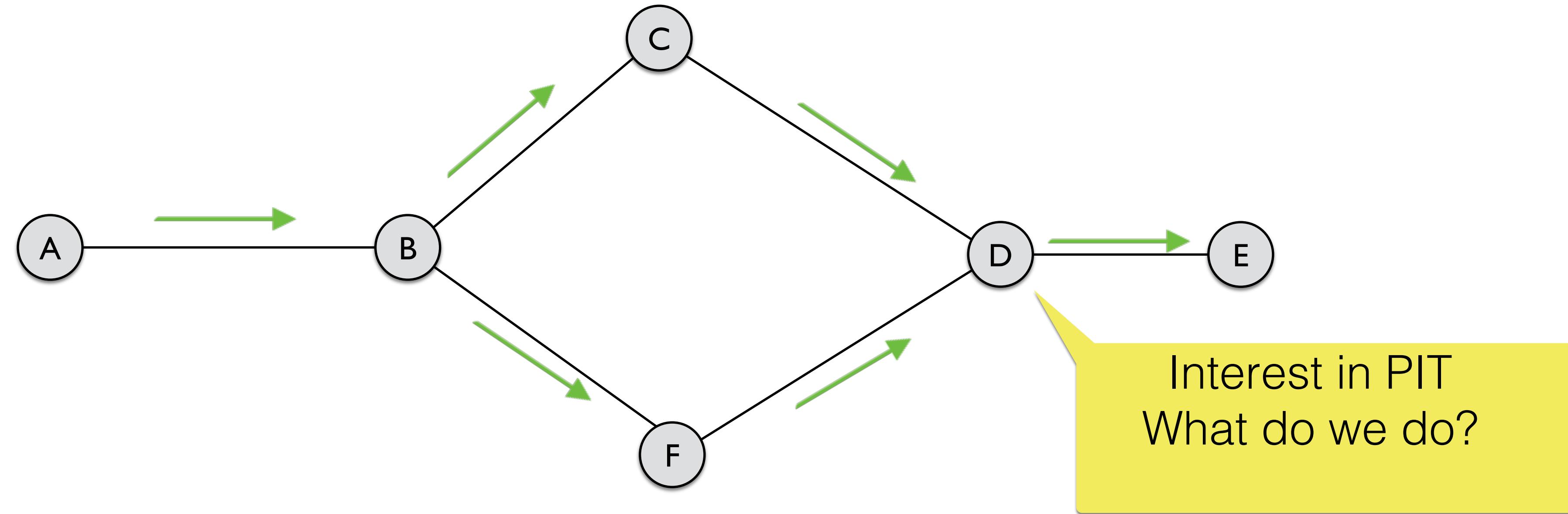


Nonces and loops



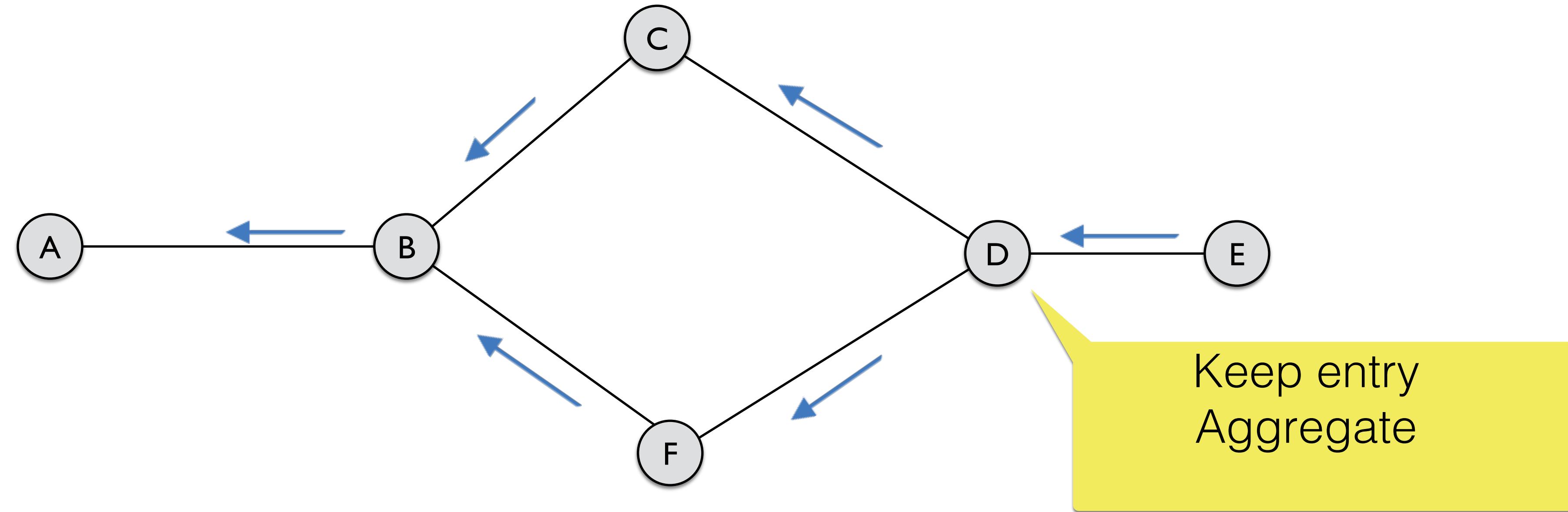
Need hop-limit to stop loop.
Nonces dont help

Nonces and multi-copy



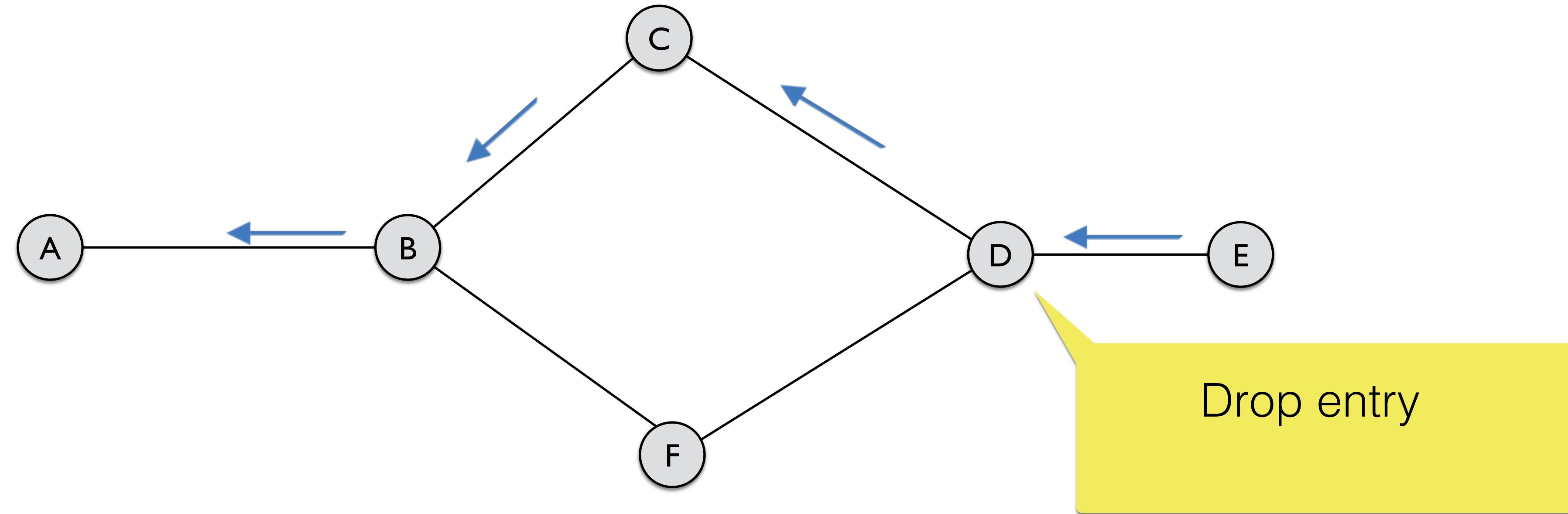
Node B sends multiple copies of interest

Nonces and multi-copy



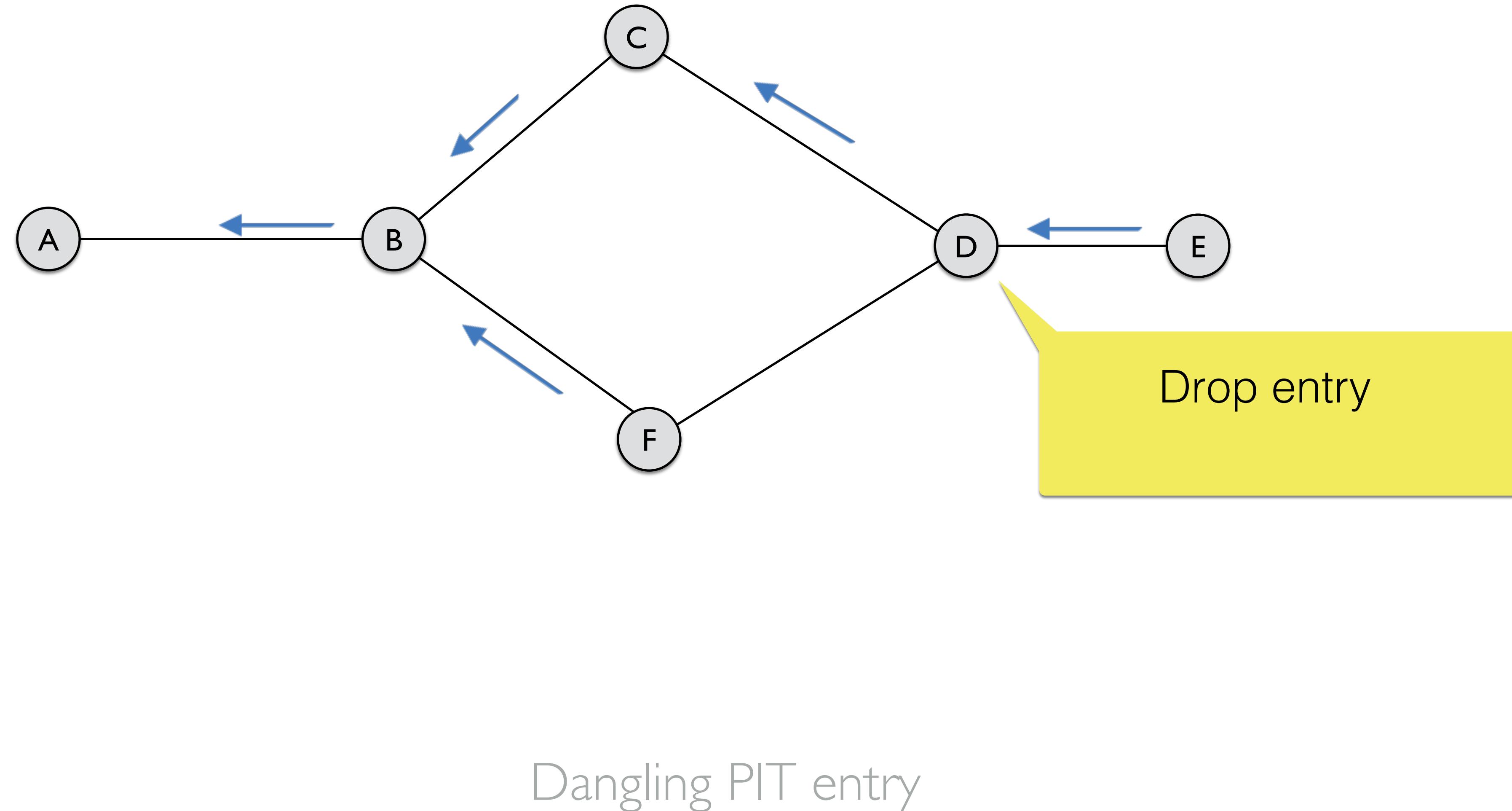
Nonce not needed to keep entry

Nonces and multi-copy

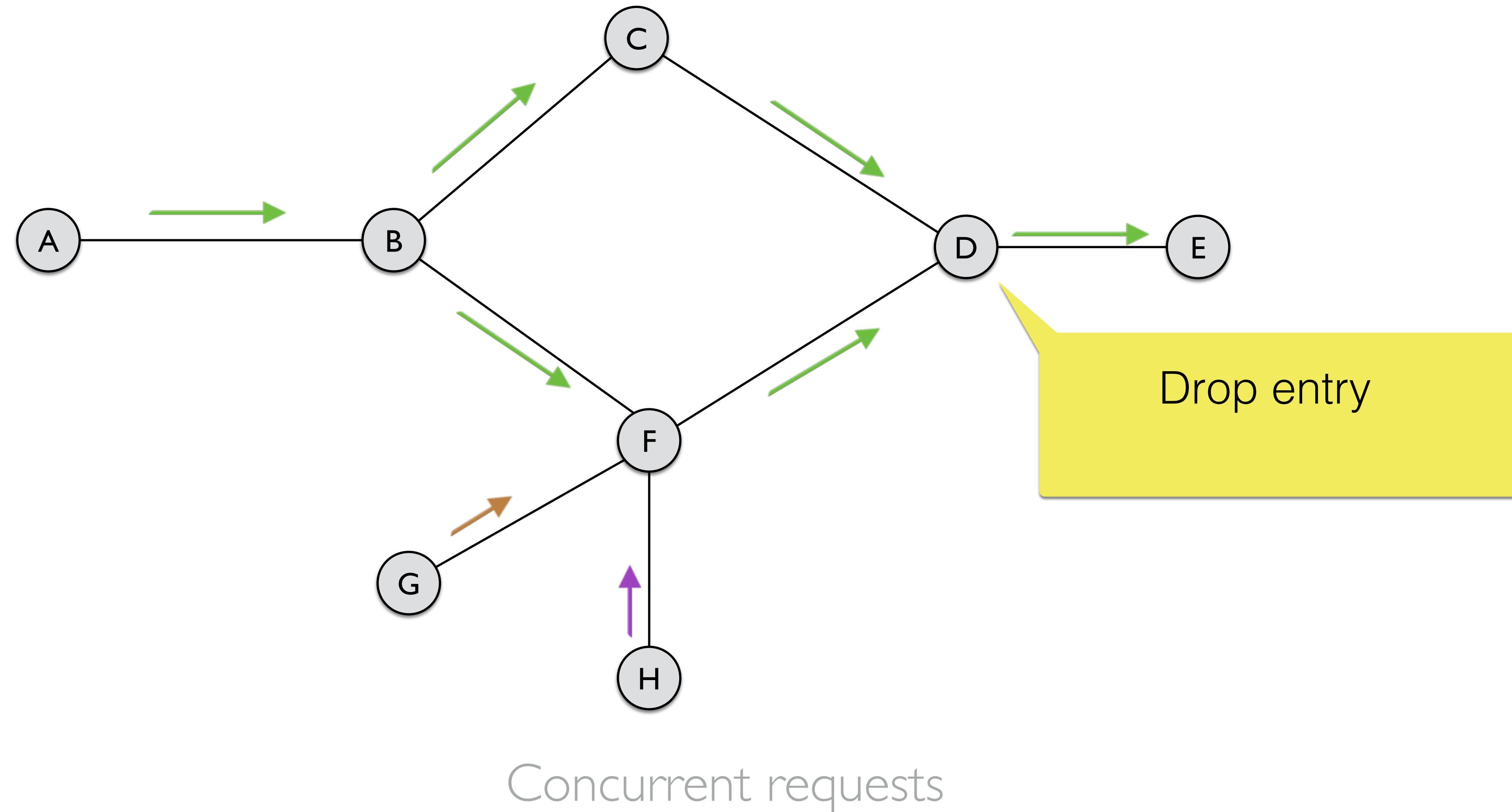


Nonce detects copy

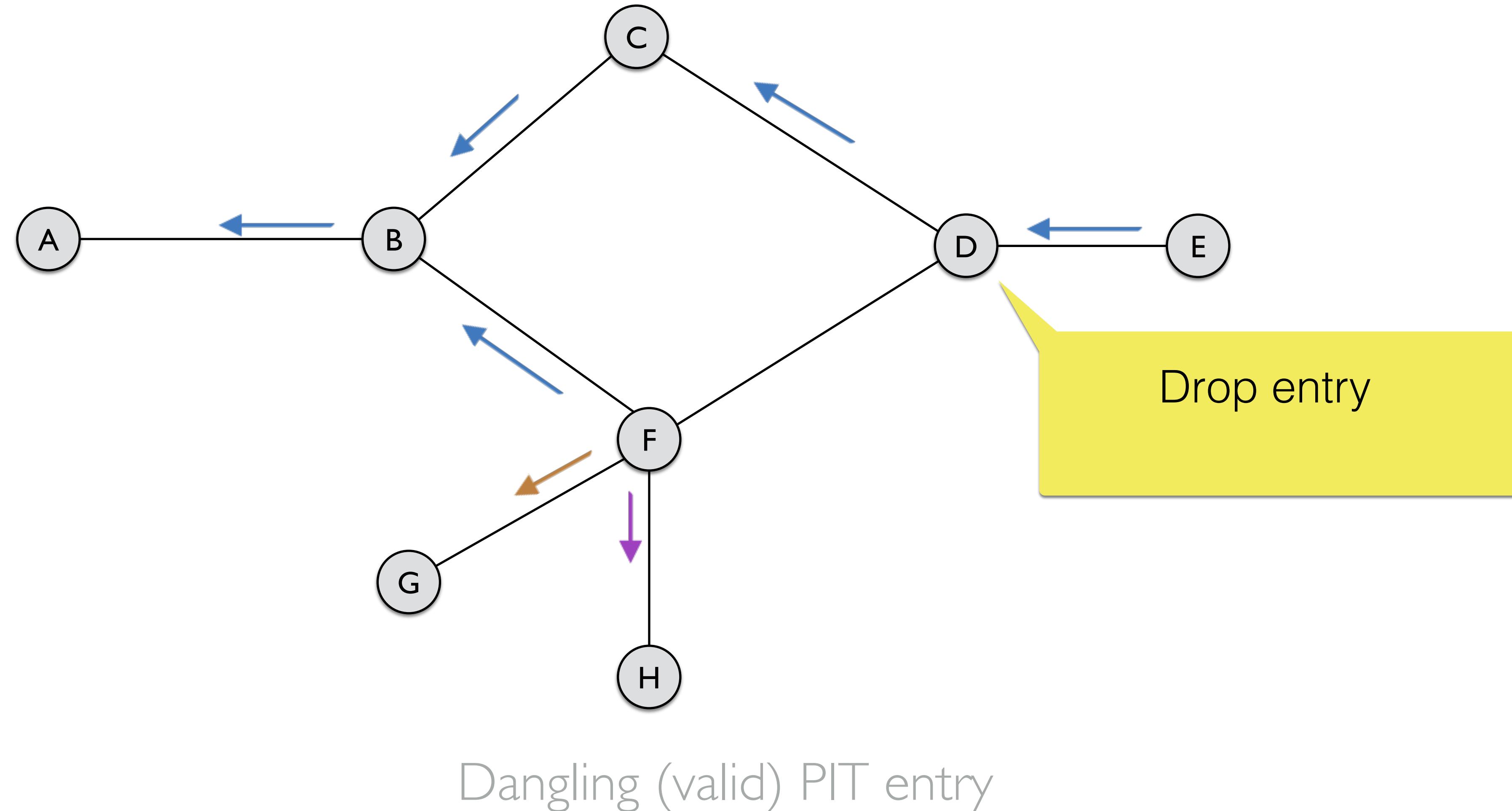
Nonces and multi-copy



Nonces and multi-copy



Nonces and multi-copy



Loop halting - Why change

Less overhead

No need to carry large nonce in packet

No need to keep nonces at router (large at fast speed)

PIT takes care of most loops

PIT halts loops, hop-limit is a stop-gap

Nonce-for-loops break aggregation

Interests can't be aggregated

(if node can treat same nonce interests as equal)

Interest Payload

CCN I.x

Interest:

name = /store/cart/id=1234/checkout

Interests can carry payload

payload = abcdefg<1k component>xyz

Interest Payload

CCN 0.x

Interest:
name = /store/cart/abcdefg...
...<|k component>xyz/checkout

CCN 1.x

Interest:
name = /store/cart/id=1234/checkout
payload = abcdefg<|k component>xyz

Interest Payload - Why change

Less processing at routers

No need to parse long names all the time

Less storage at routers

No need to keep large names at routers

Less traffic overhead

No need to carry copy of state back in the response

Protocol Separation

Core protocol runs everywhere

Protocol specified individually

Separation of concerns

Modularity

Discovery

Chunking

Versioning

Core Messaging

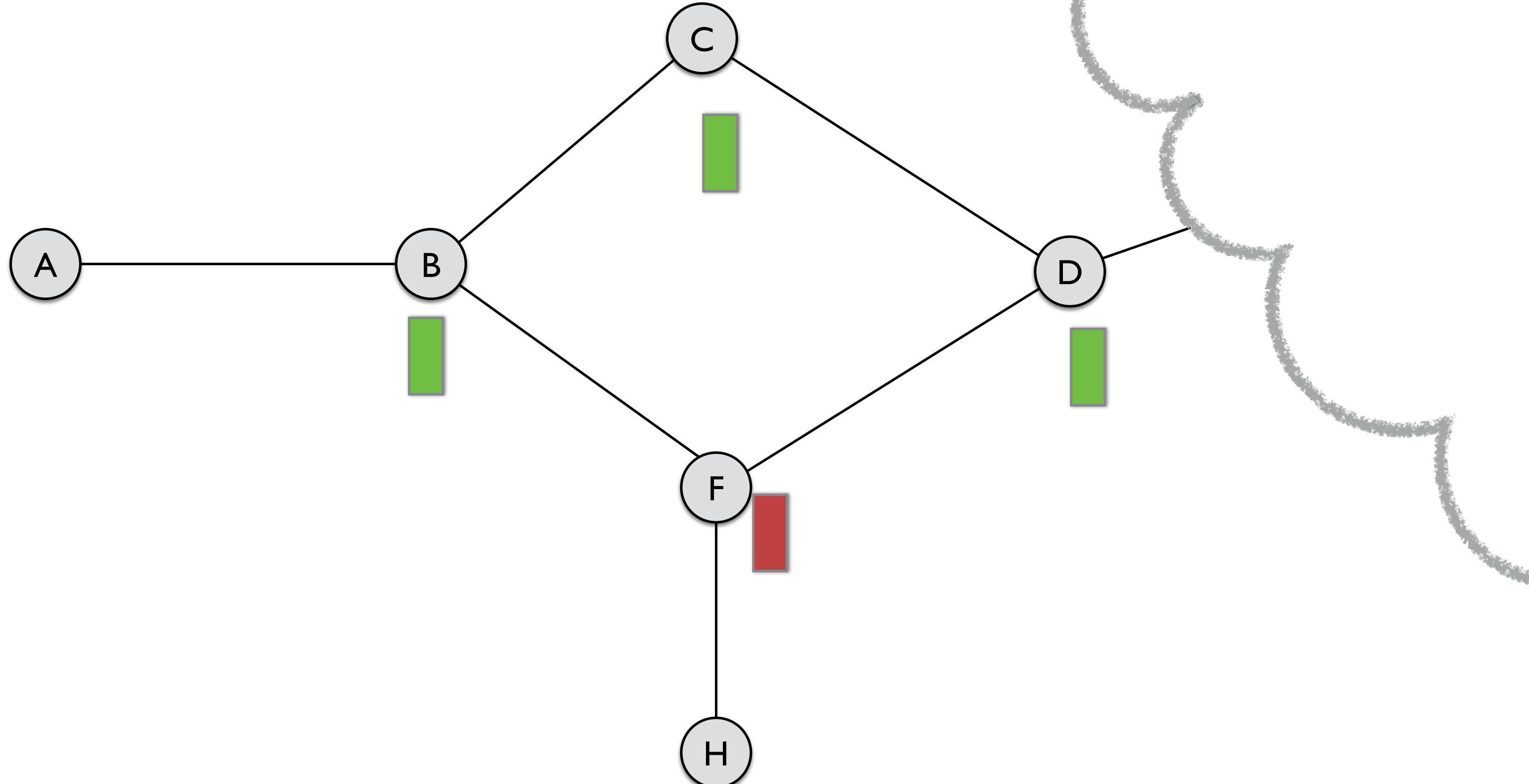
Framing

Caching

Universal

Optional

Restricted

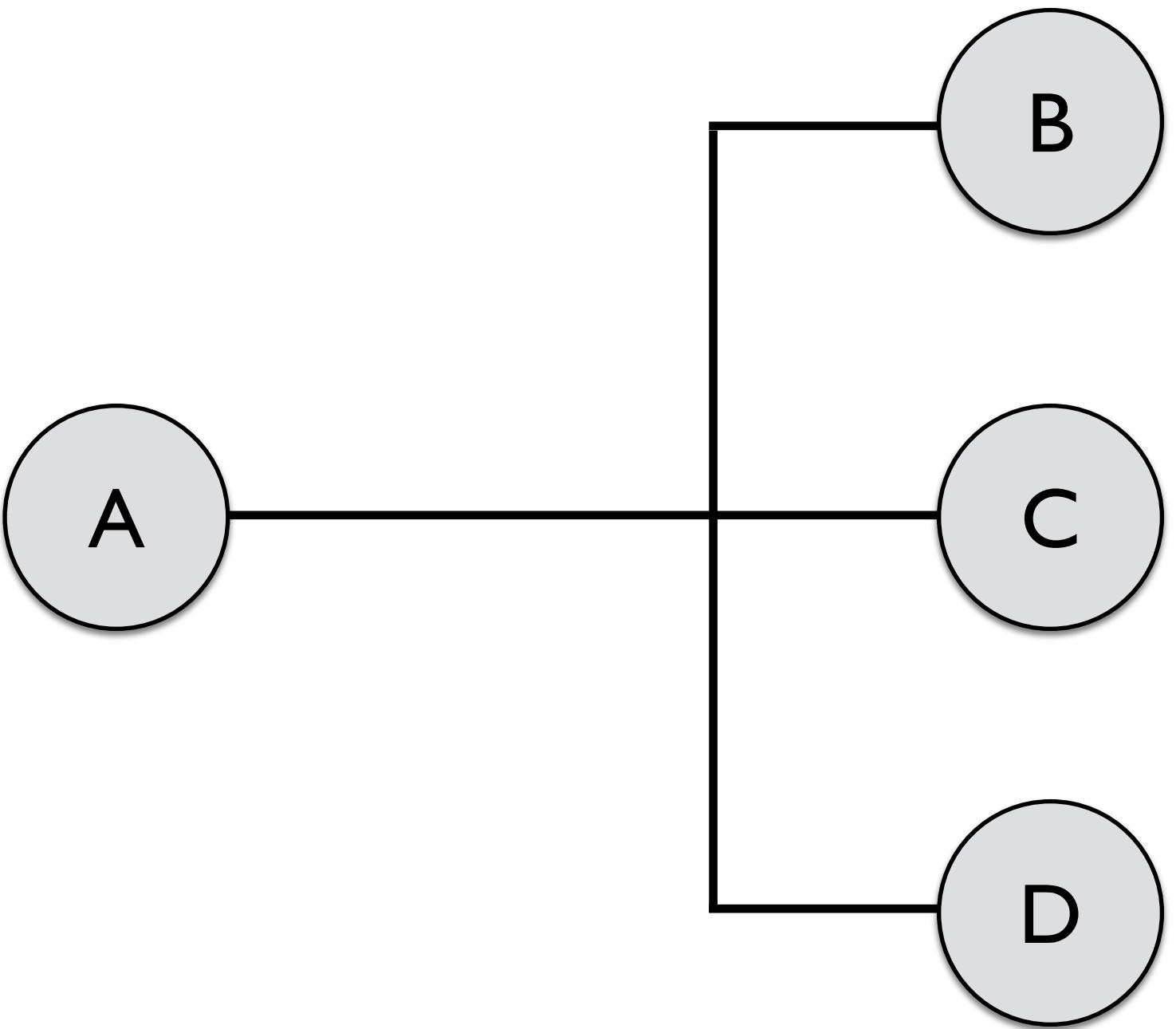


Layer 2

Next hop

Mapping to link layer (ARP)

Fragmentation



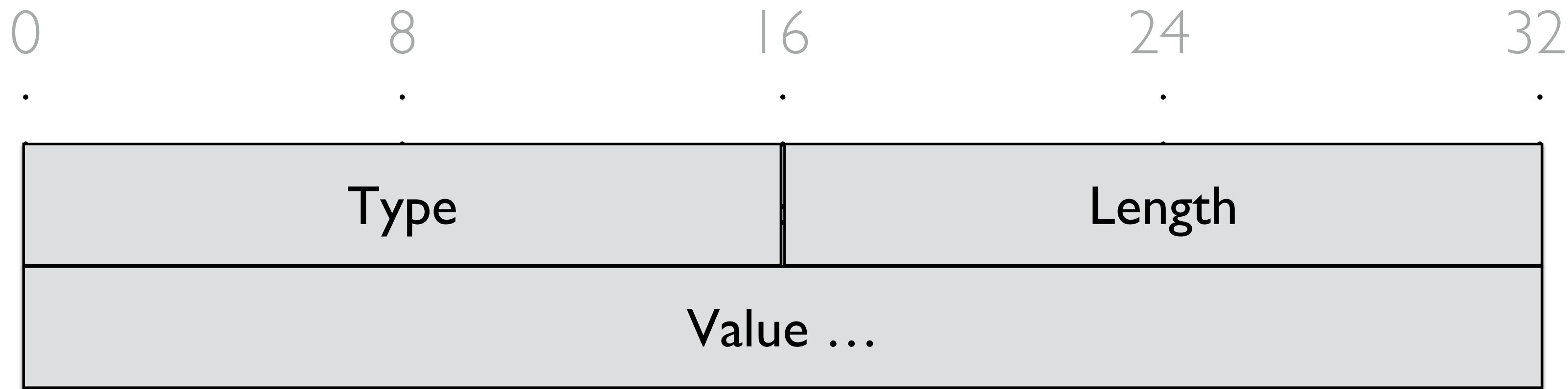
CCN - Encoding

CCN - Packets

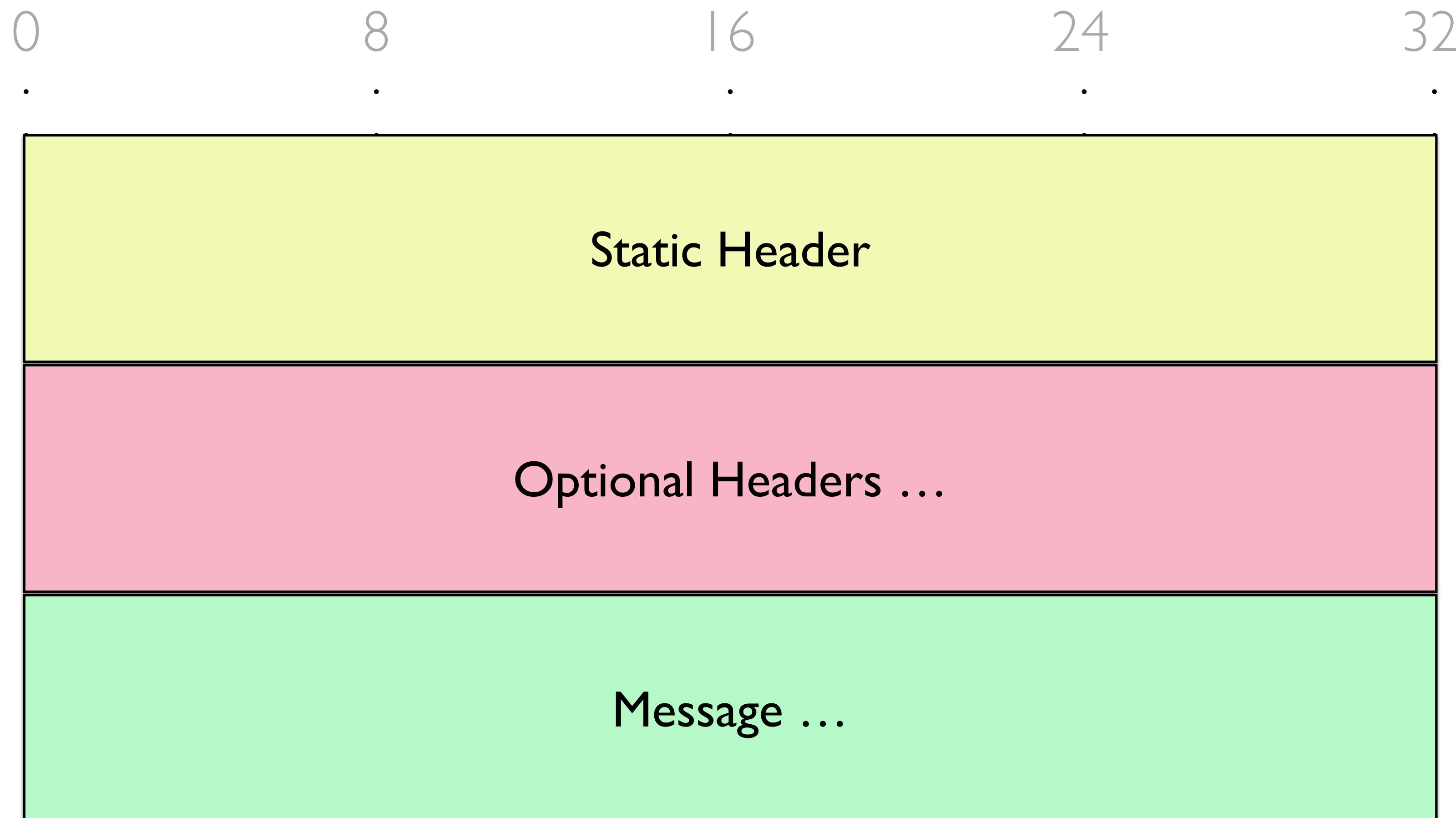
Type-Length-Value (TLV) encoding



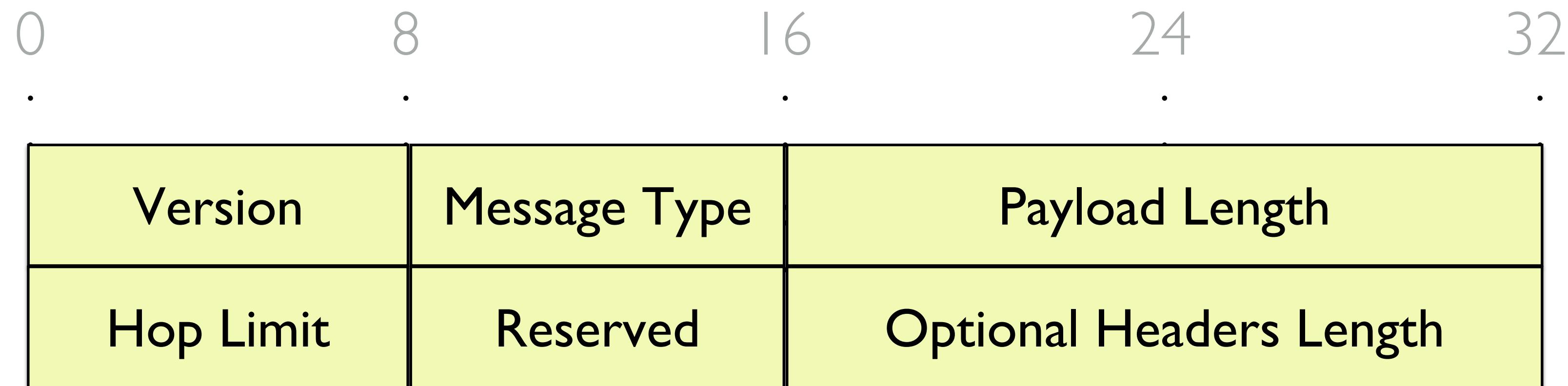
2 x 2 encoding



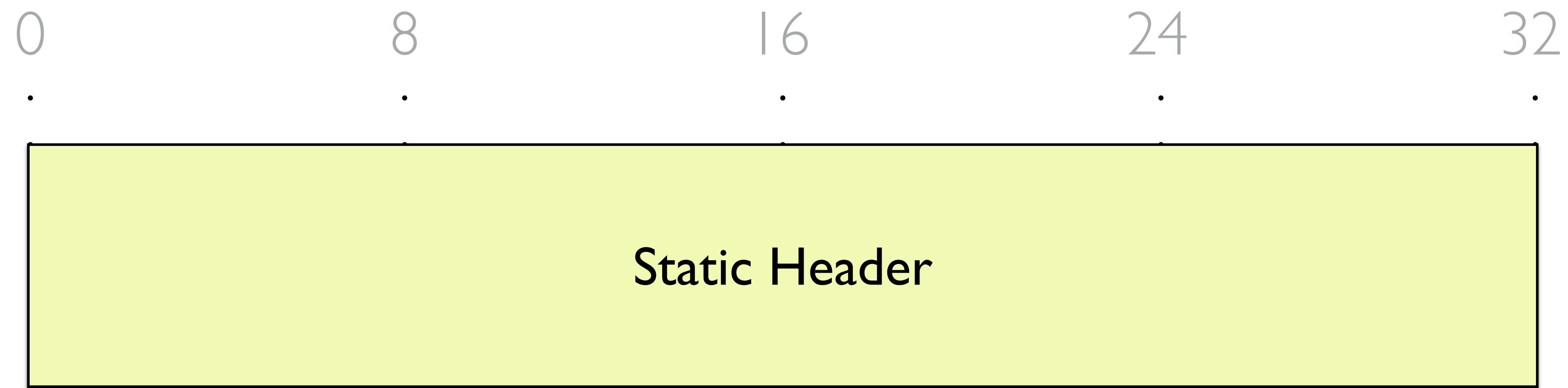
CCN Packet Structure



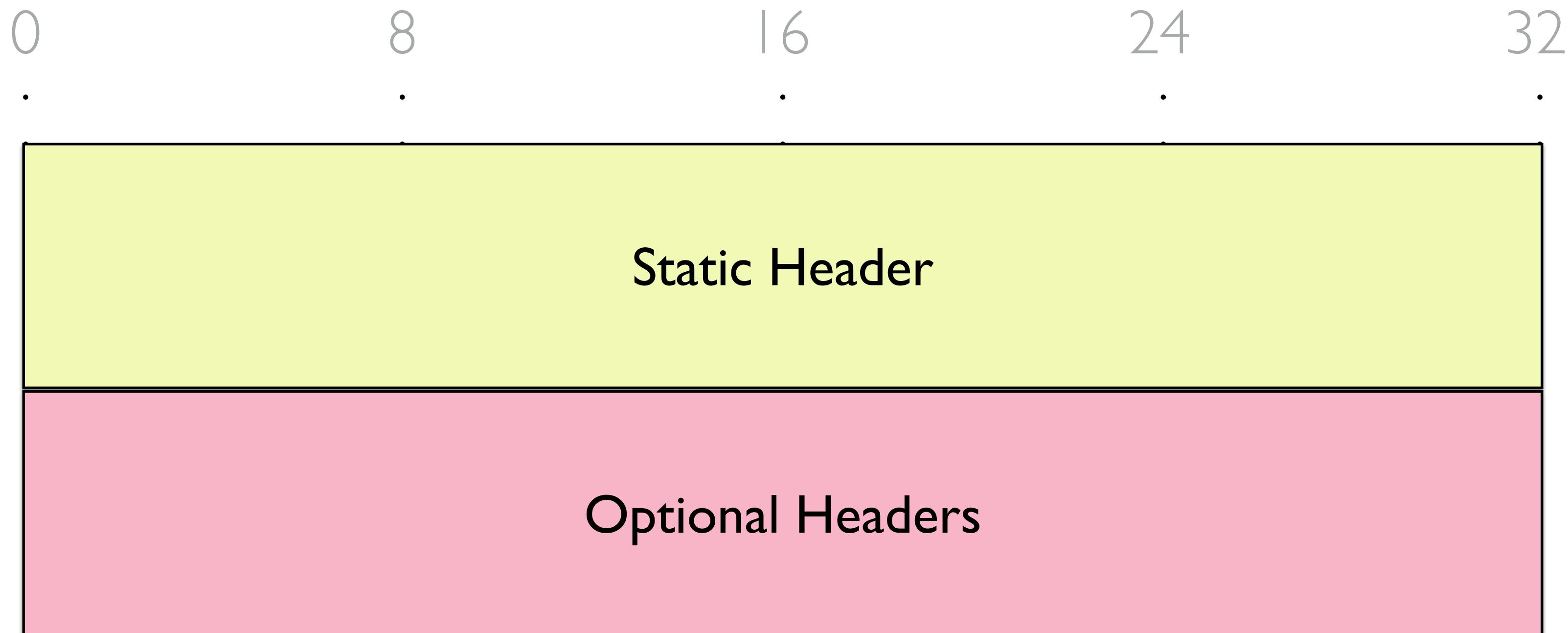
Static Header



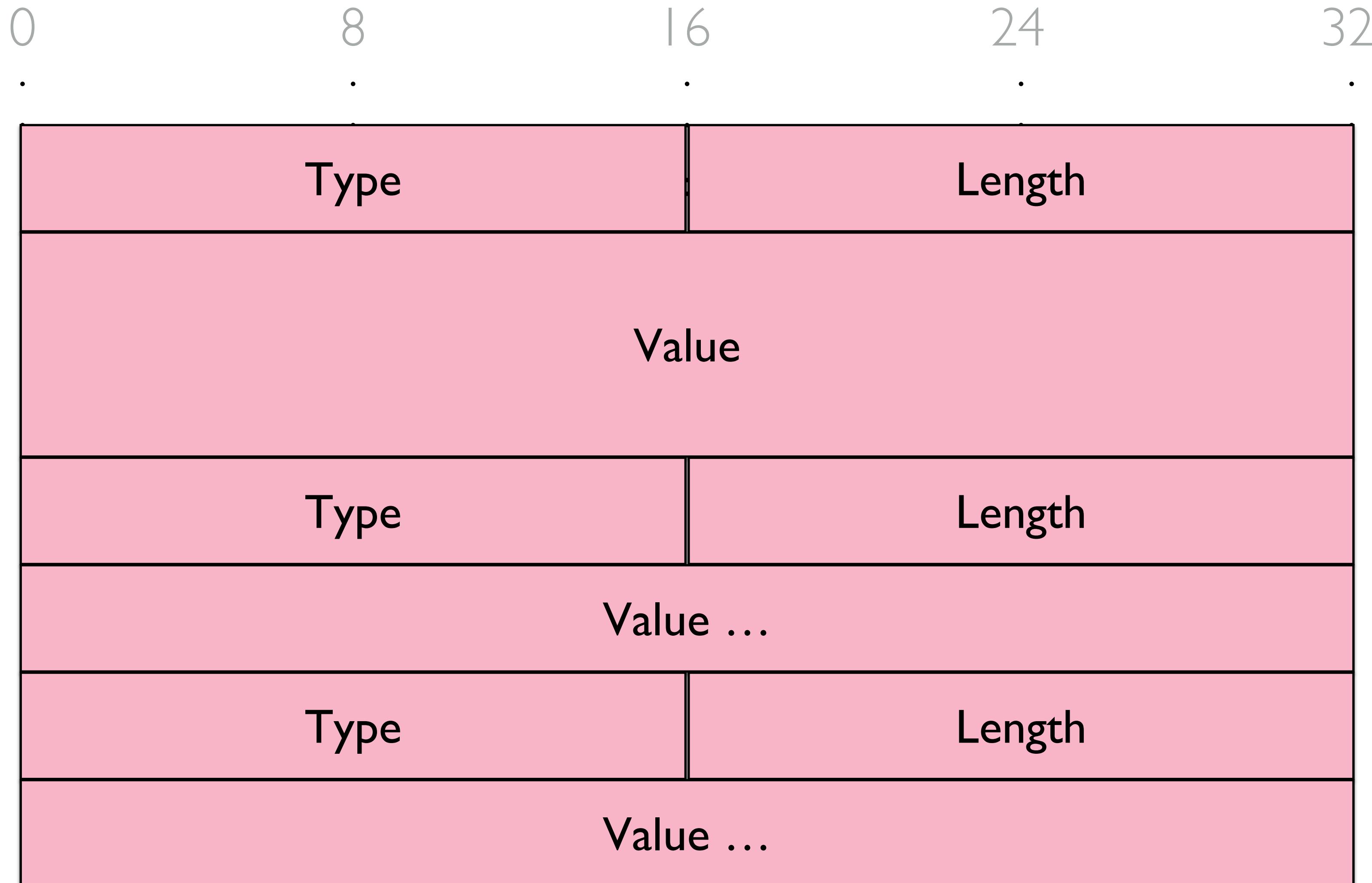
Static Header



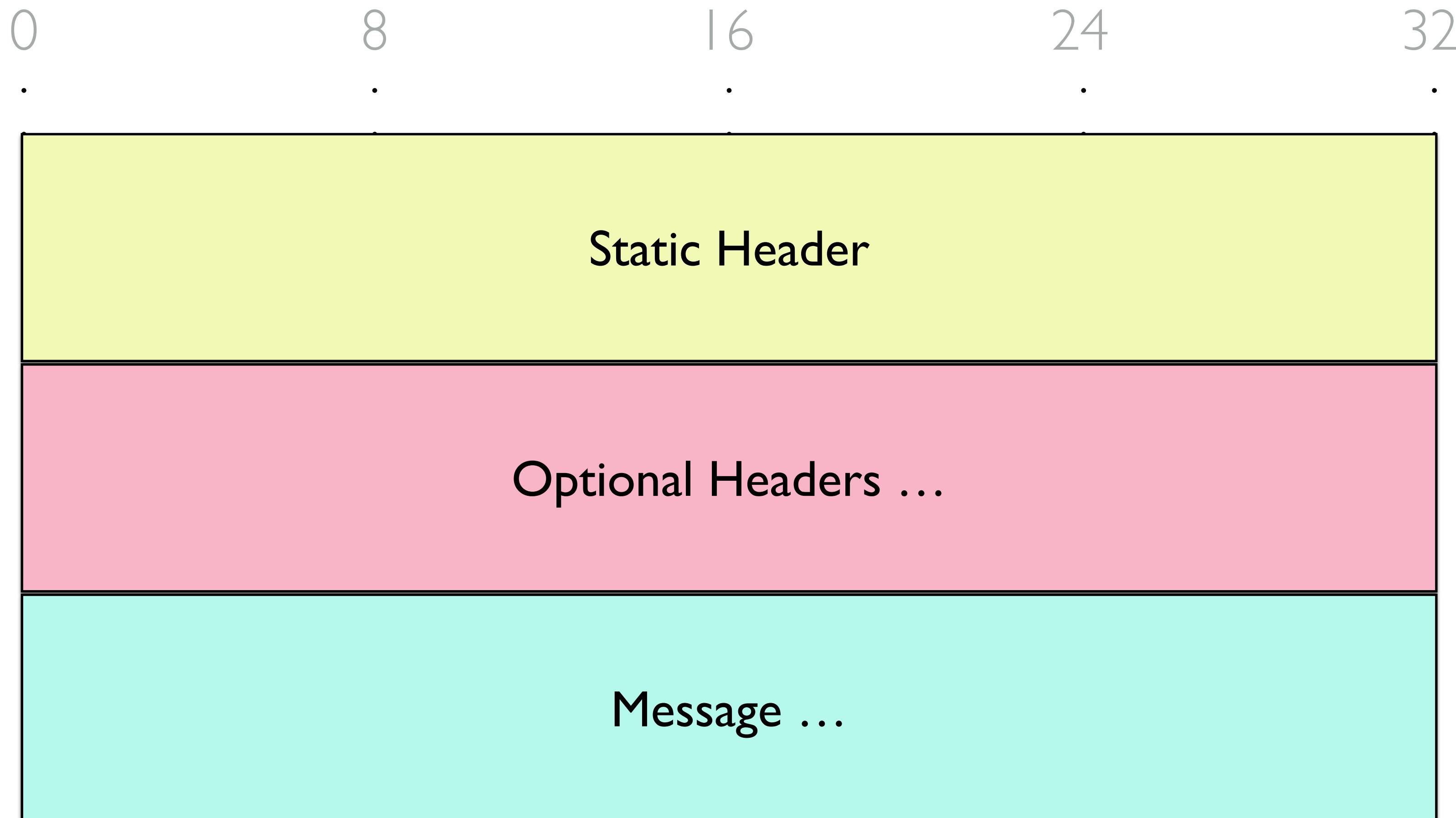
CCN Headers



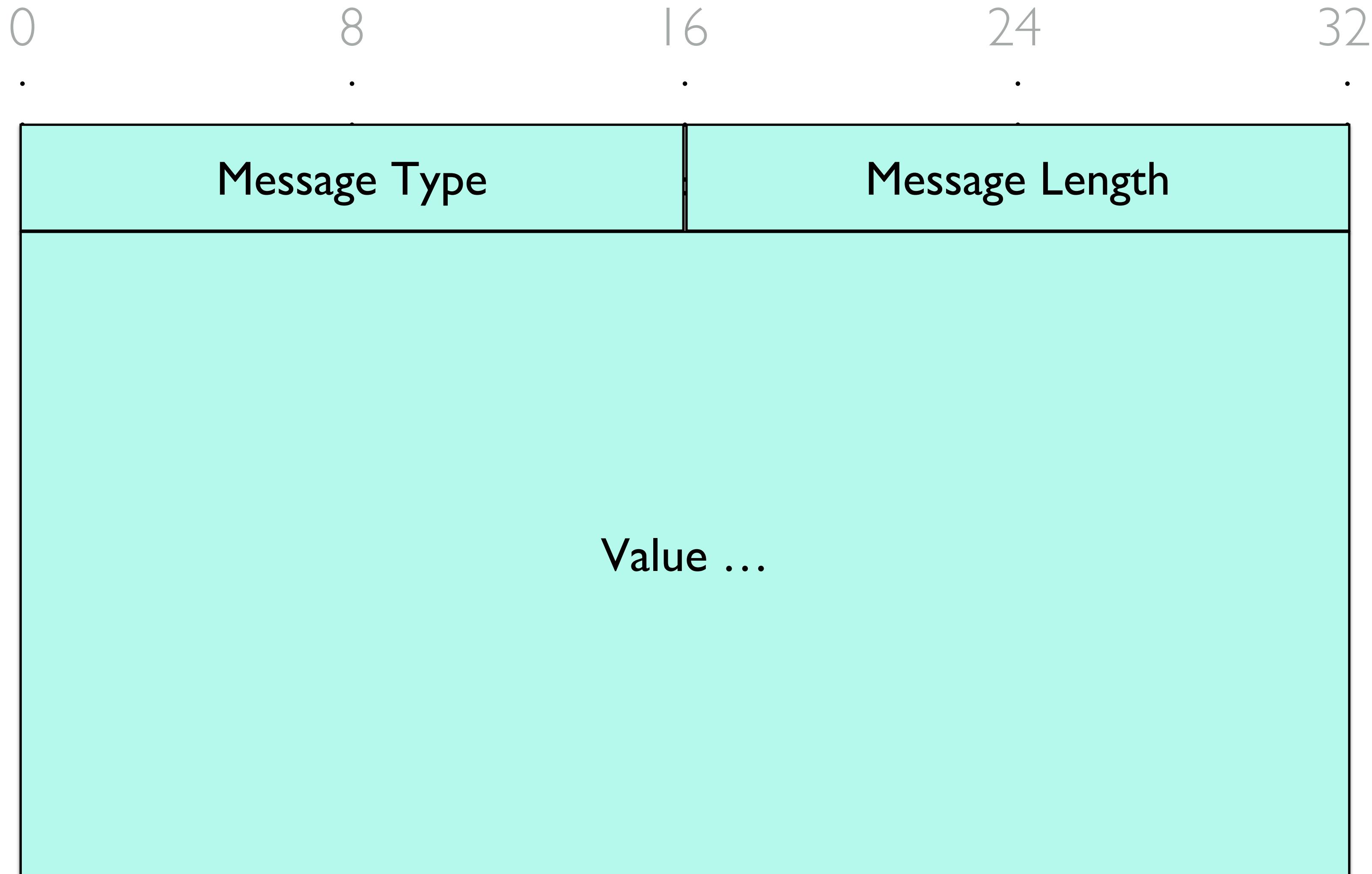
Optional Headers



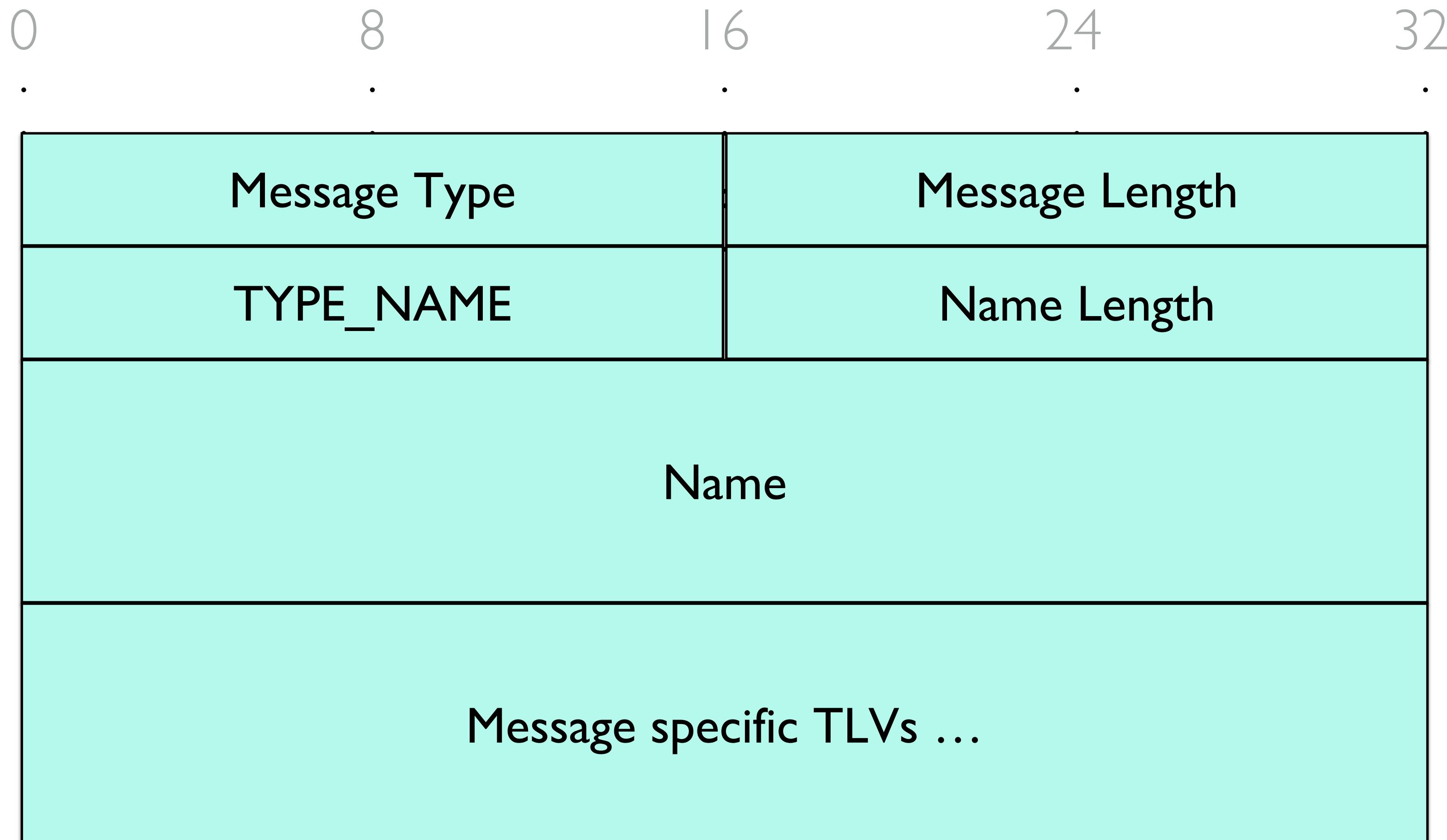
CCN Packet Structure



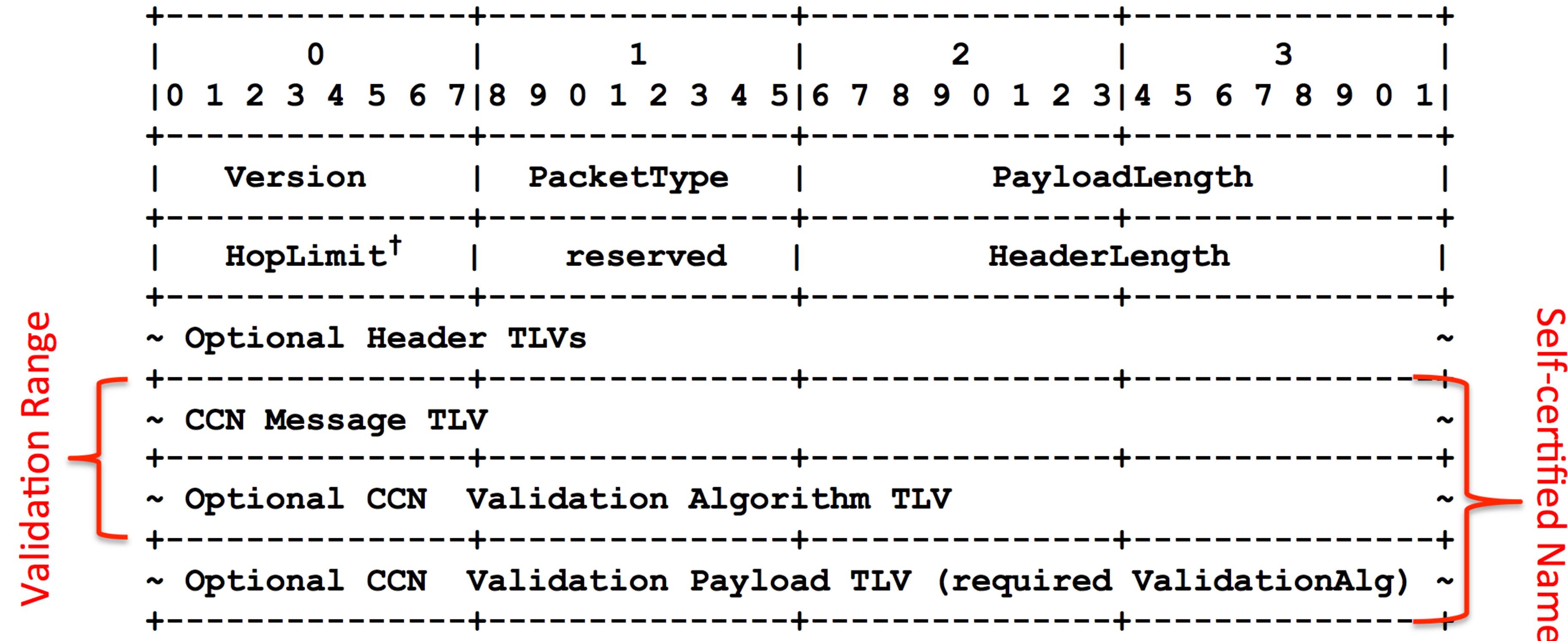
CCN Message



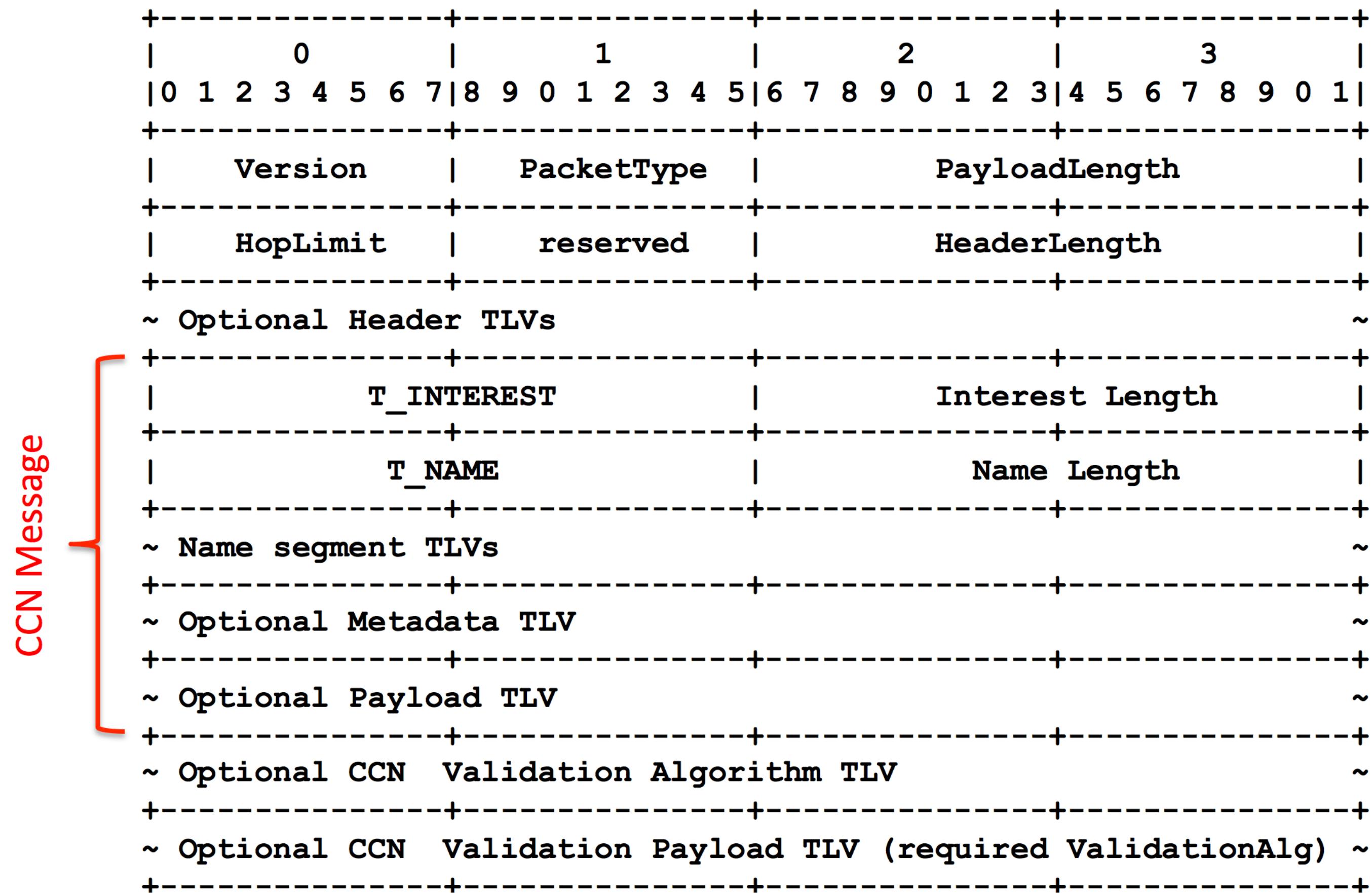
CCN Message



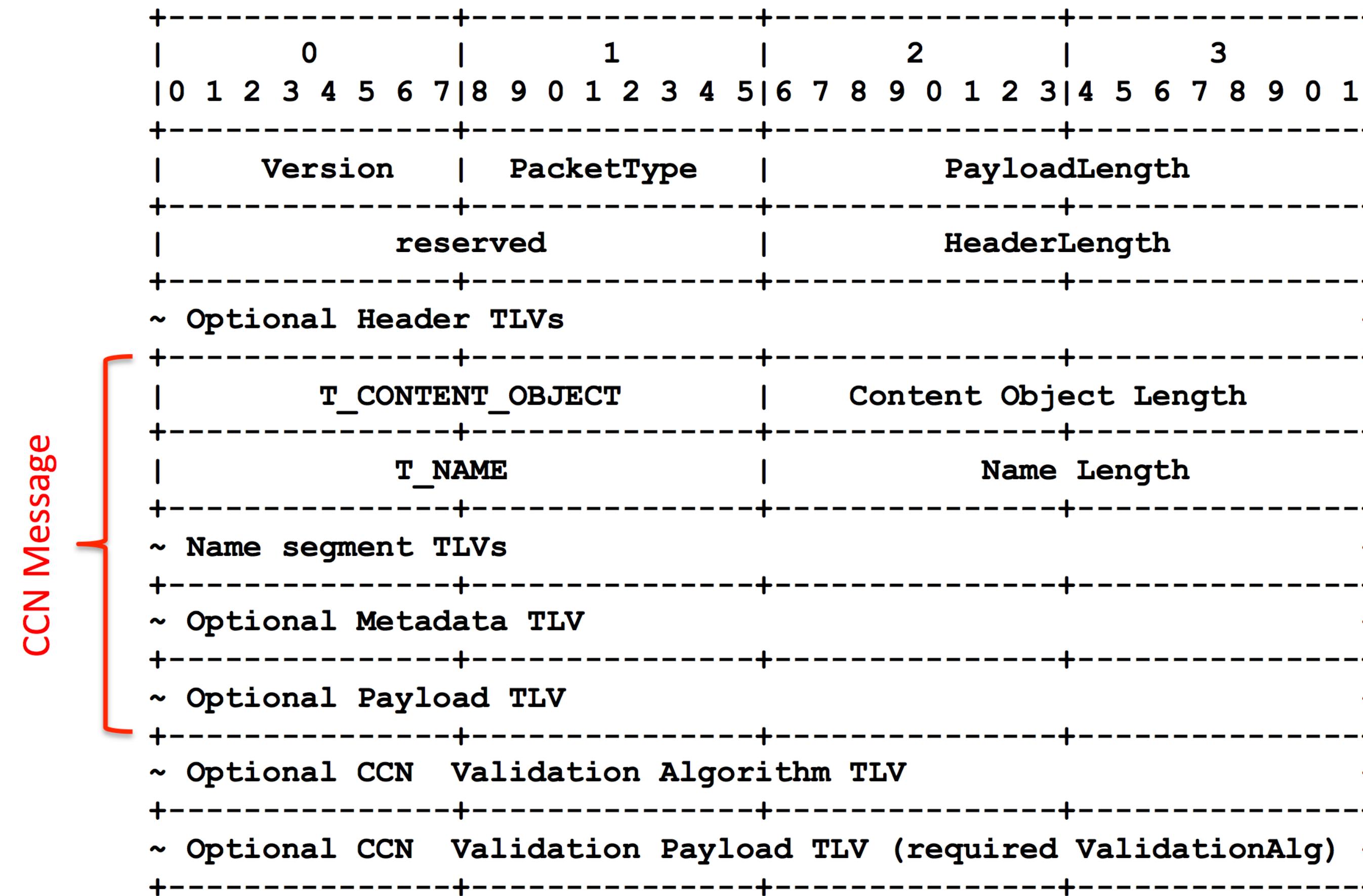
CCN Packet



Interest Message Packet



Content Object Packet



CCN - Names

CCN Name structure

/segment1/segment2/segment3

/parc/ccnx/presentation

/parc/ccnx/presentation/serial=2/

/parc/ccnx/presentation/serial=2/chunk=6

/parc/ccnx/presentation/serial=2/app:slide=80

CCN name segment structure

label=segment

CCN name segment structure

/ccn/presentation

/name=ccn/name=presentation

Default label = name

CCN name segment structure

chunk=6
serial=6

Chunk is used for chunked objects
Serial is used for versioning

CCN name segment structure

app:<param>=2014

app:year=2014

Application specific labels used by apps

Labeled Content Identifiers (CCN URIs)

lci:/parc/ccn/spec/serial=3/chunk=2

Labeled Content Identifiers

lci:/parc/ccn/spec/serial=3/chunk=2

lci:/name=parc/name=ccn/name=spec/serial=3/chunk=2

Labeled Content Identifiers

lci:/parc/ccn/spec/serial=3/chunk=2

lci:/name=parc/name=ccn/name=spec/serial=3/chunk=2

lci:/N=parc/N=ccn/N=spec/V=3/C=2

Labeled Content Identifiers

lci:/parc/ccn/spec/serial=3/chunk=2

lci:/name=parc/name=ccn/name=spec/serial=3/chunk=2

lci:/N=parc/N=ccn/N=spec/V=3/C=2

lci:/0x0010=parc/0x0010=ccn/0x0010=spec/0x00A1=3/0x00A3=2

Labeled Content Identifiers

lci:/parc/pics/spec/app:year=2014

Labeled Content Identifiers

lci:/parc/pics/spec/app:year=2014

lci:/name=parc/name=pics/app:year=2014/app:03=bob

Labeled Content Identifiers

lci:/parc/pics/spec/app:year=2014

lci:/name=parc/name=pics/app:year=2014/app:03=bob

lci:/0x0010=parc/0x0010=pics/0xFF01=2014/0xFF03=bob

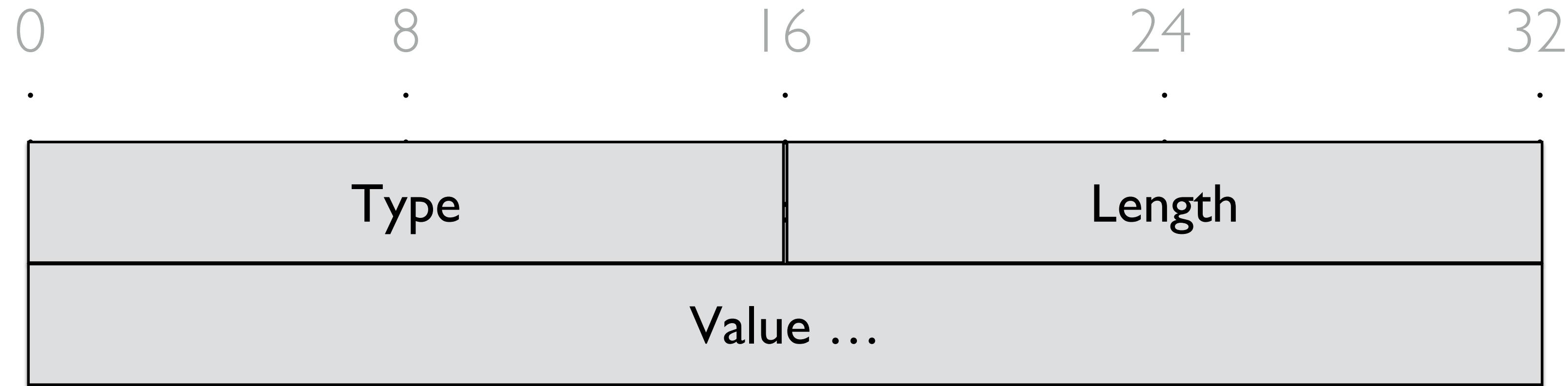
lci:/ scheme based off of URIs (RFC 3986)

```
LS-URI          = scheme ":" ls-hier-part [ "?" ls-query ]
                  ["#" fragment]
ls-hier-part    = ["// authority] ls-path-absolute
ls-path-absolute = "/" [ ls-segment *( "/" ls-segment ) ]
ls-segment      = lpv-segment / v-segment
lpv-segment     = label [ ":" param ] "=" s-value
v-segment        = s-value
label            = alpha-t / num-t
param            = alpha-t / num-t
s-value          = *(s-pchar)

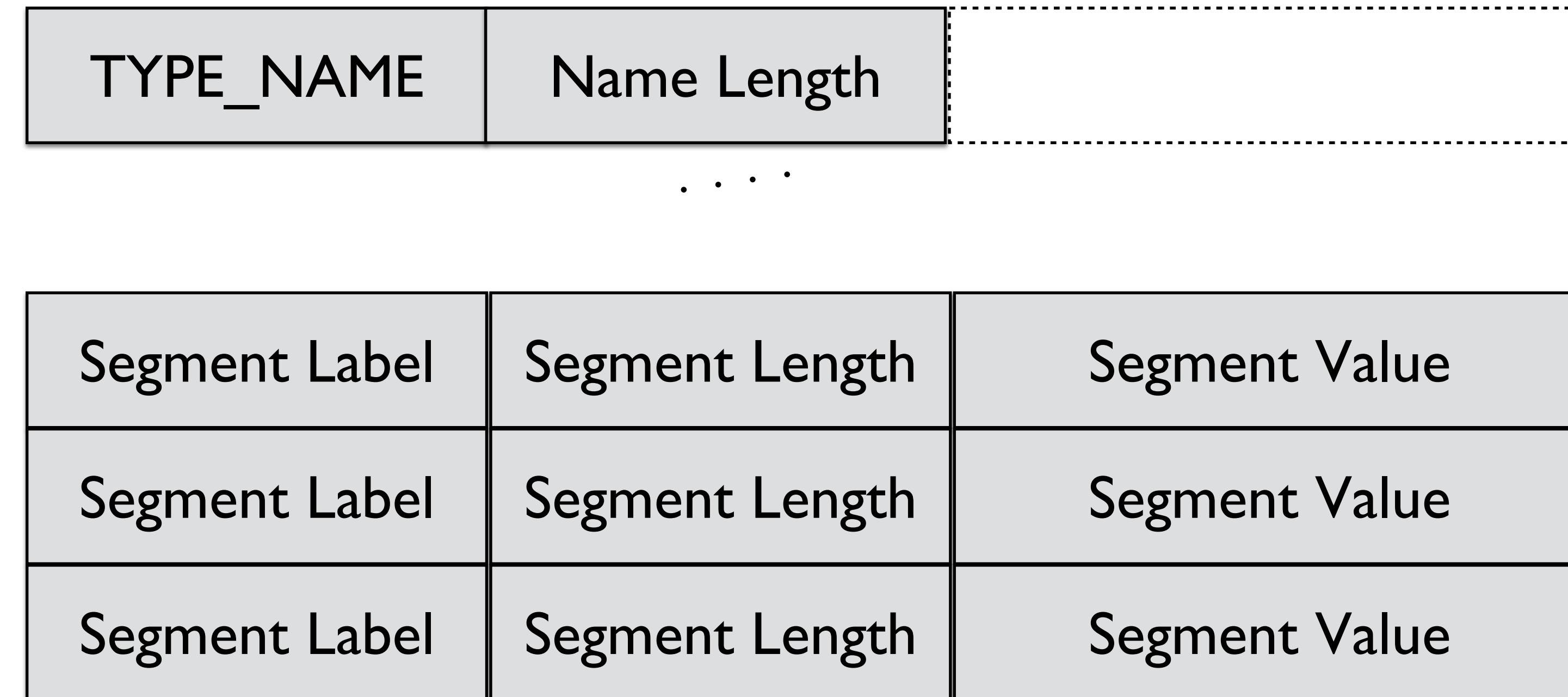
ls-query         = *1 ( lpv-component / v-component
                      *( "&" (lpv-component / v-component) ) )
lpv-component   = label [ ":" param ] "=" q-value
v-component     = q-value
q-value          = *(q-pchar)

alpha-t          = ALPHA *(ALPHA / DIGIT)
num-t            = dec-t / hex-t
dec-t            = 1*(DIGIT)
hex-t            = "0x" 1*(HEXDIG)
ls-pchar          = unreserved / pct-encoded / ls-sub-delims
s-pchar          = ls-pchar / ":" / "@" / "&"
q-pchar          = ls-pchar / ":" / "@" / "/"
ls-sub-delims   = !" / $" / '"' / "(" / ")"
                  / "*" / "+" / "," / ";"
```

CCN name TLV encoding

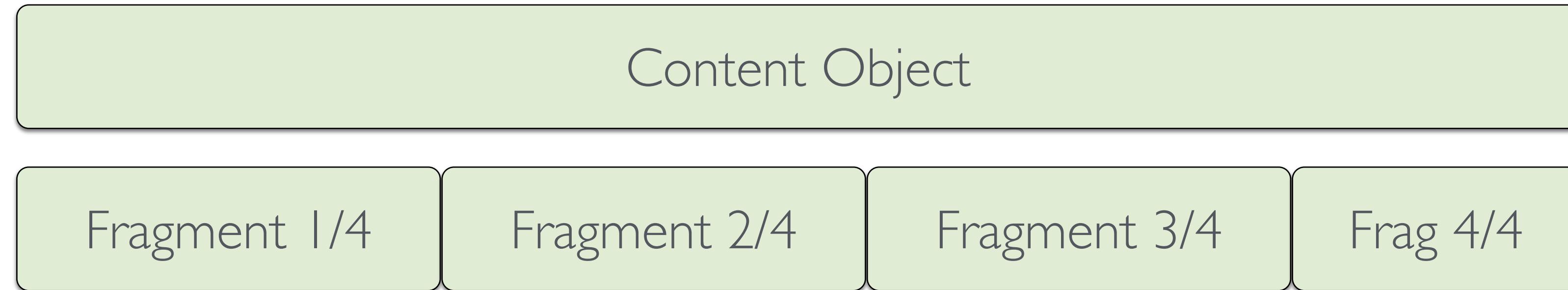


CCN name TLV encoding



CCN - Fragmentation

Fragmentation



Content Objects can be up to 64KB in size

To transmit Content Objects over network links with smaller MTU (Maximum Transmission Unit) size we need to fragment Content Objects

Fragmentation Types

Hop-by-Hop

Link specific, not part of spec

Mid-to-End

Not supported

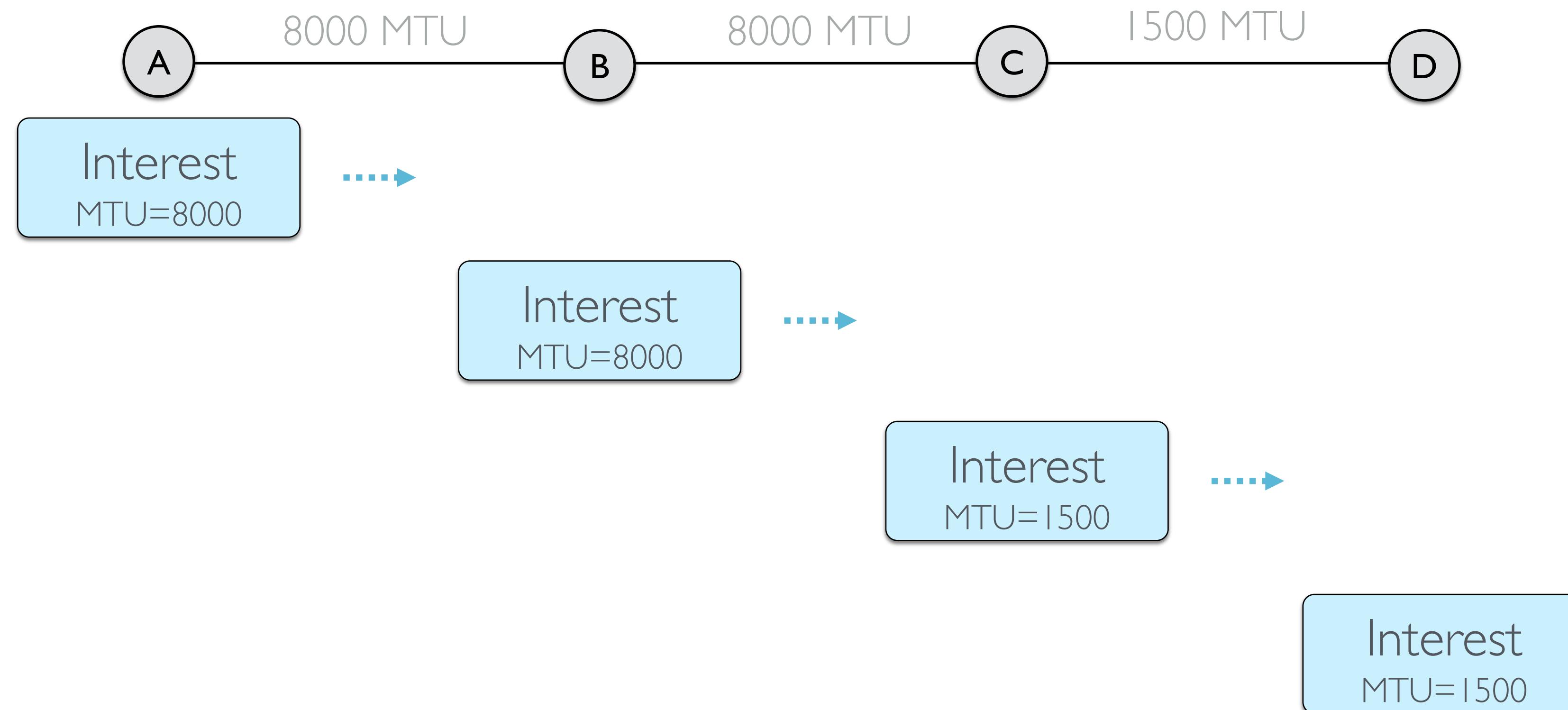
End-to-End

Interest discovers MTU

None

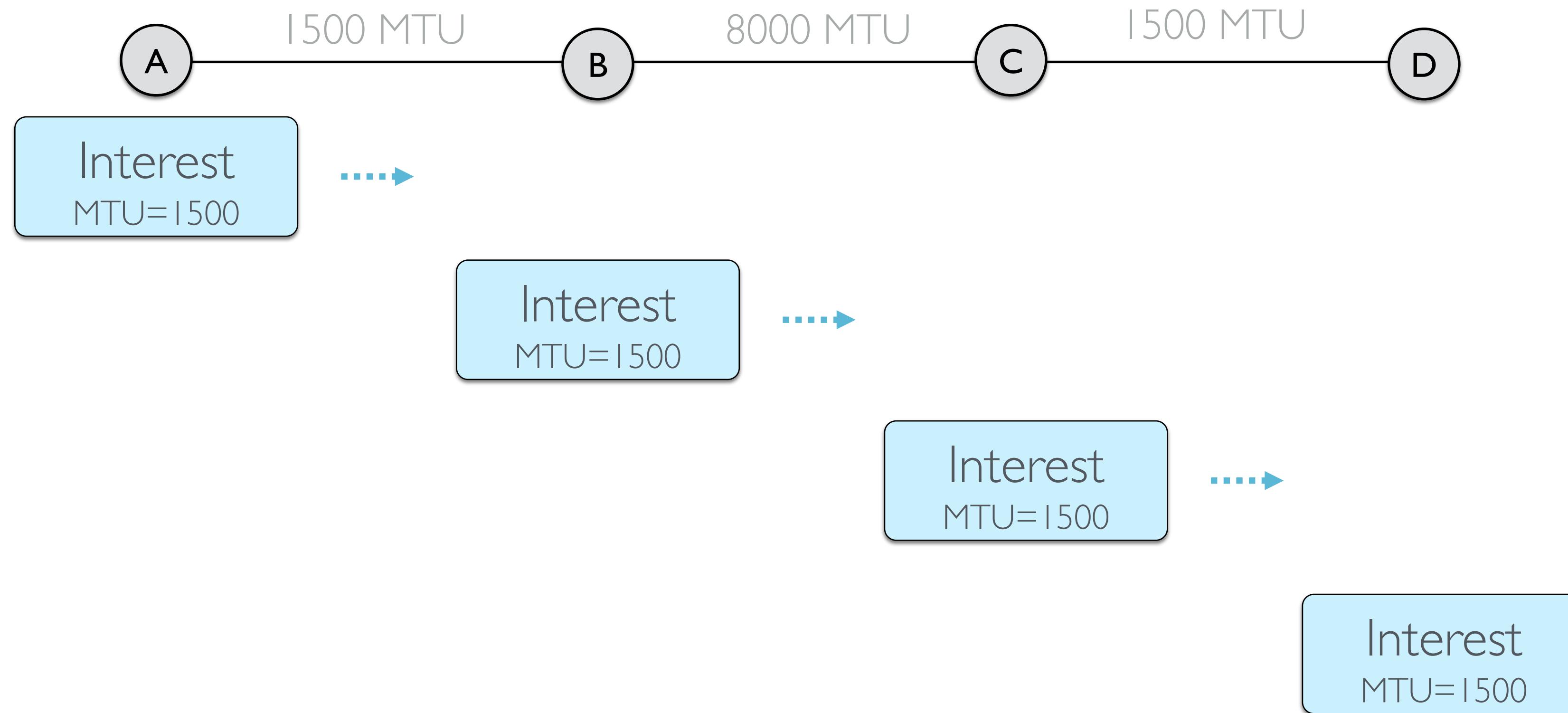
Preferred mode, potentially more efficient

End-to-End Fragmentation



Interests discover the MTU

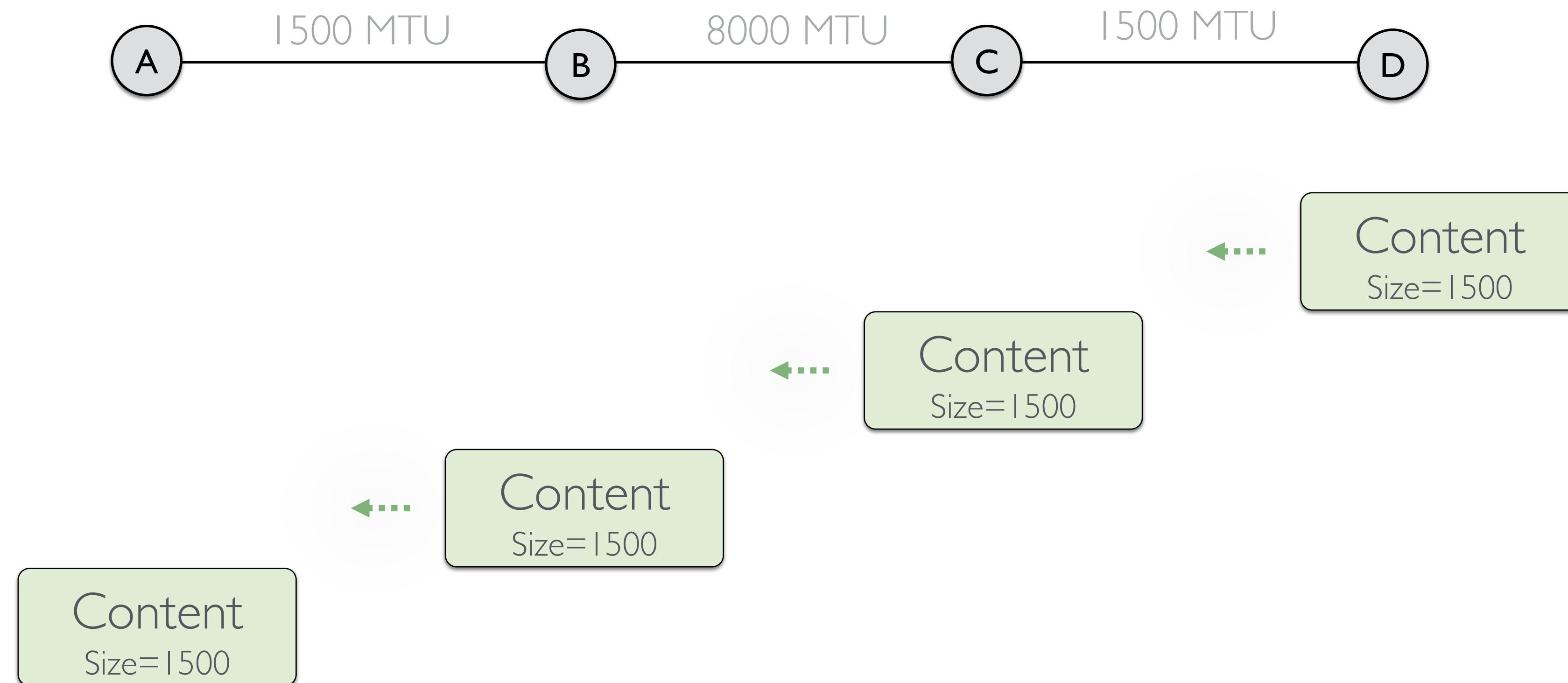
End-to-End Fragmentation



Interests are fragmented to minimum MTU

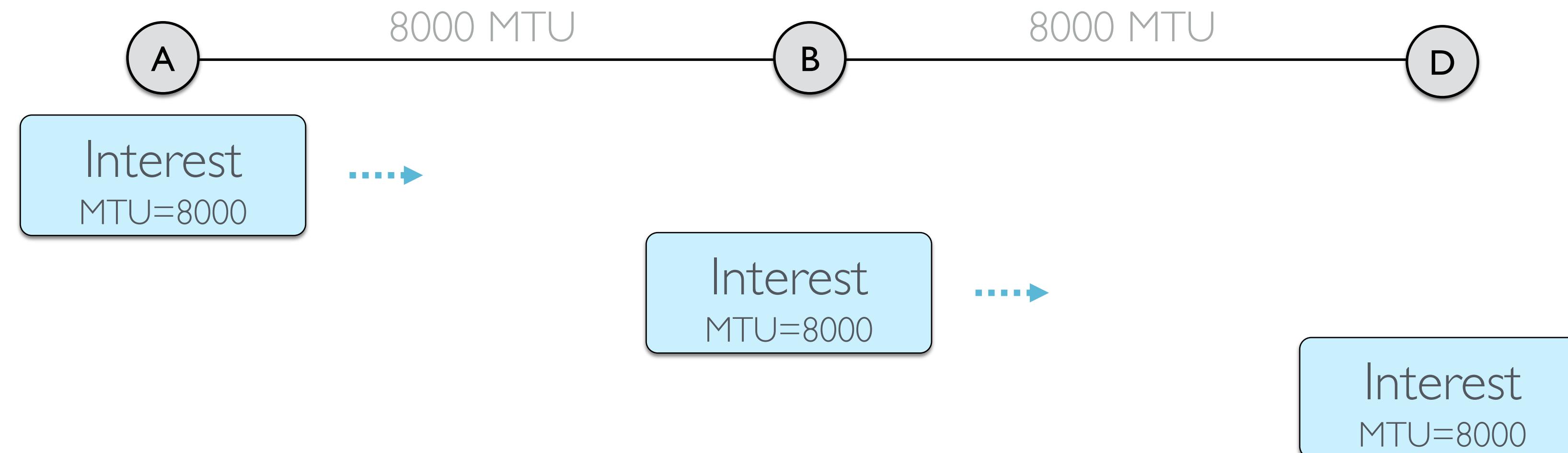
First Interest fragment must be routable

End-to-End Fragmentation



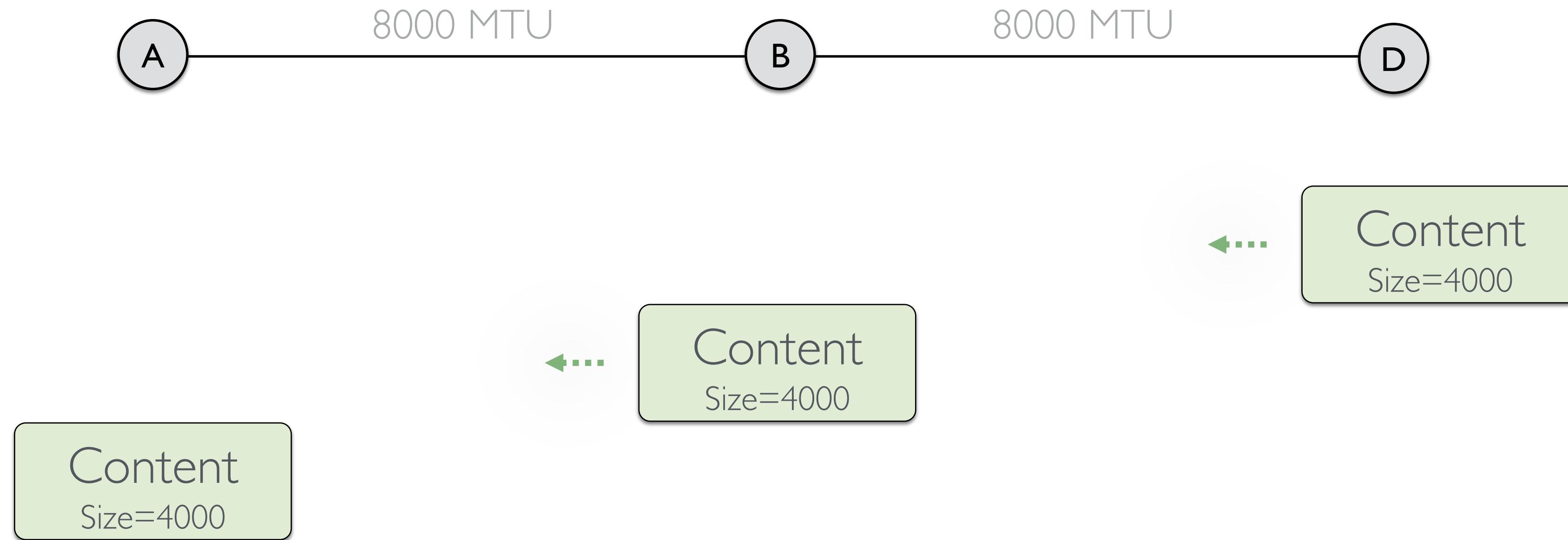
Content is fragmented to discovered MTU

End-to-End Fragmentation



Local networks might have large MTU

End-to-End Fragmentation



Reply might be fragmented smaller than MTU discovered

End-to-End Fragmentation Characteristics

Interests fragmented to minimum MTU at Requester

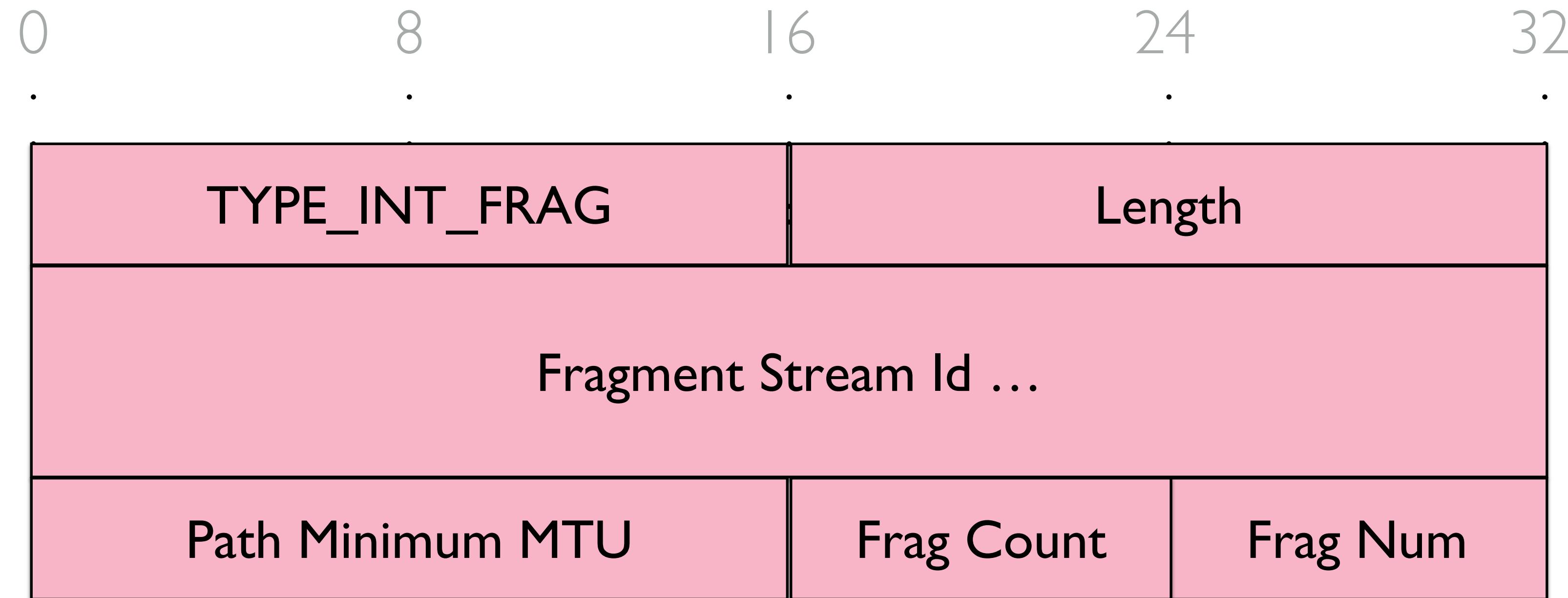
First Interest fragment must be routable

Content is fragmented to MTU size

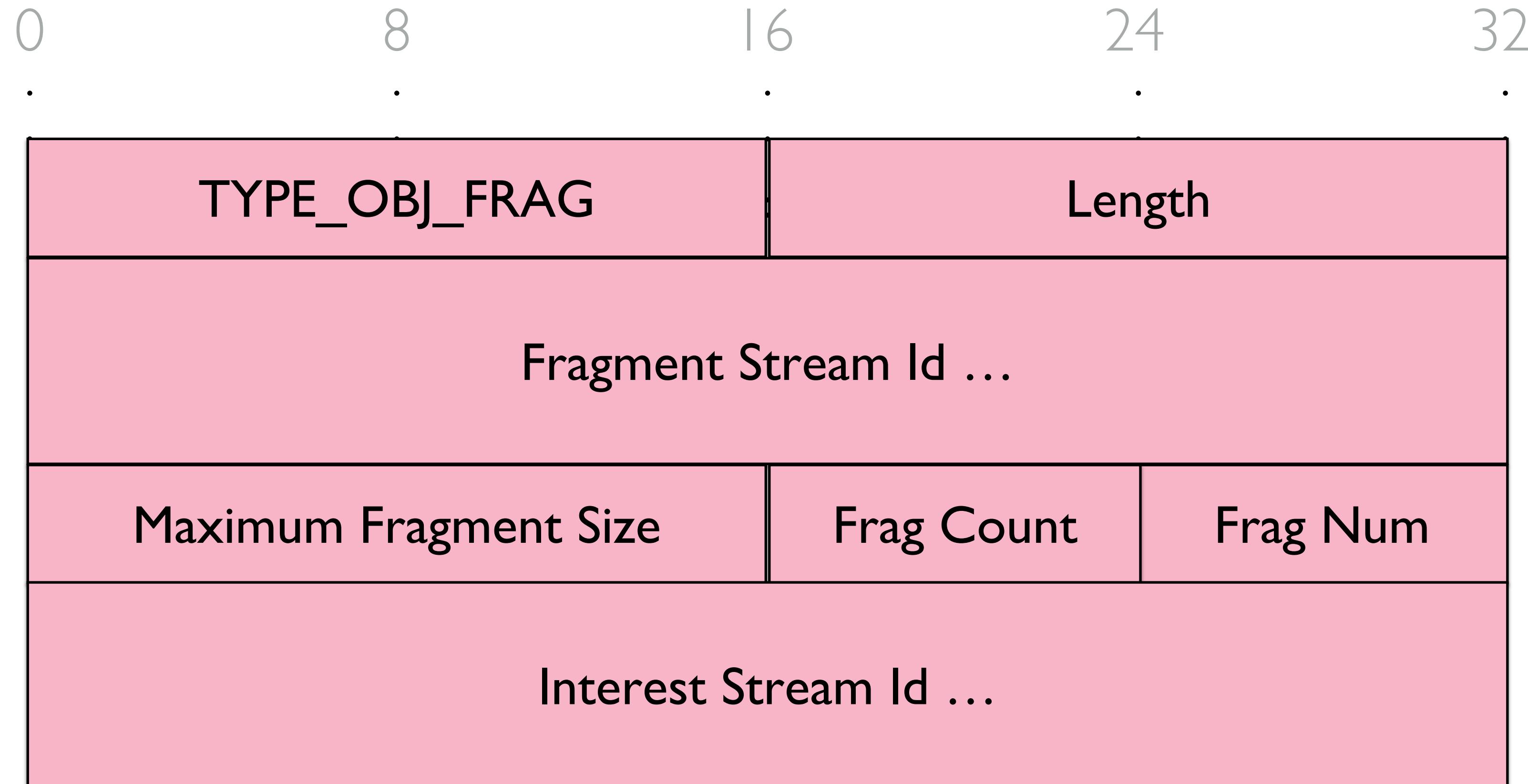
Nodes reassemble if checking Content-Object Hash

Fragments are packed (“MTU” size, except last)

Fragmentation Optional Header



Fragmentation Optional Header



End-to-End Fragmentation Summary

Cut through forwarding of interests

Cut through forwarding of content via Interest Id

Works in conjunction with Express Headers

Advanced fragmentation scheme in the works

Advanced Topics

Advanced topics

Routing (loop free)

Retransmissions

Placeless objects

Push

Trust models

Hash based forwarding

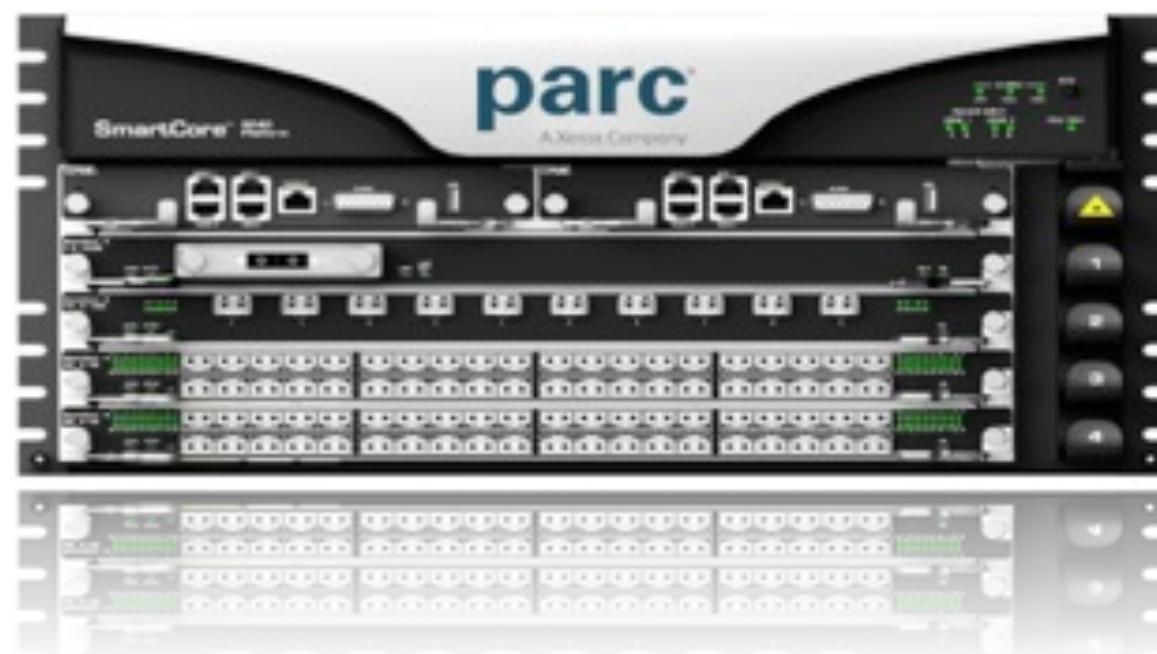
SDN

Return channel overhead

Asymmetric paths

Hardware

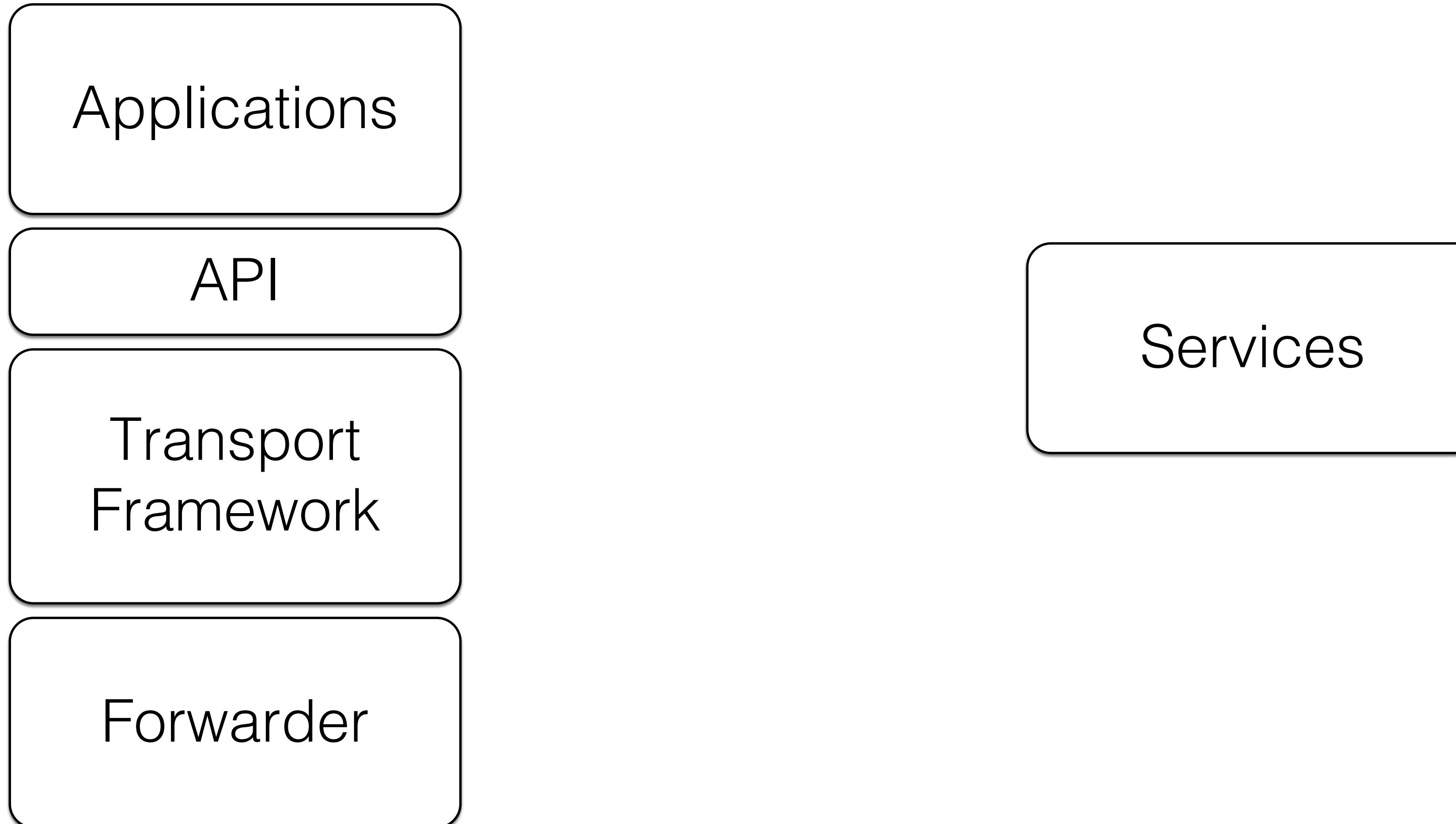
Penn
11 Terabit non-blocking fabric
14 slot chassis



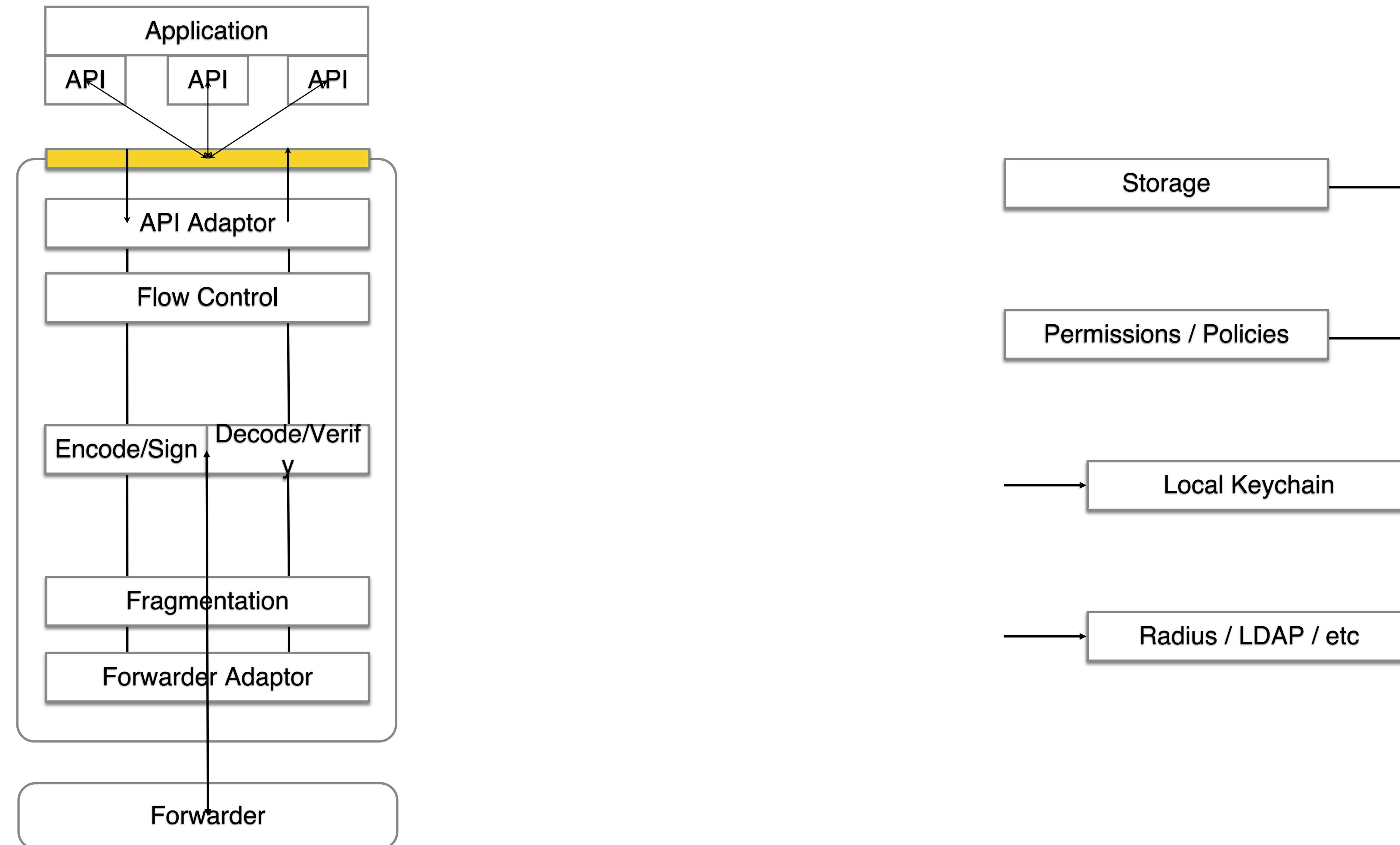
Teller
4.4 Terabit non-blocking fabric
6 slot chassis

Software Architecture

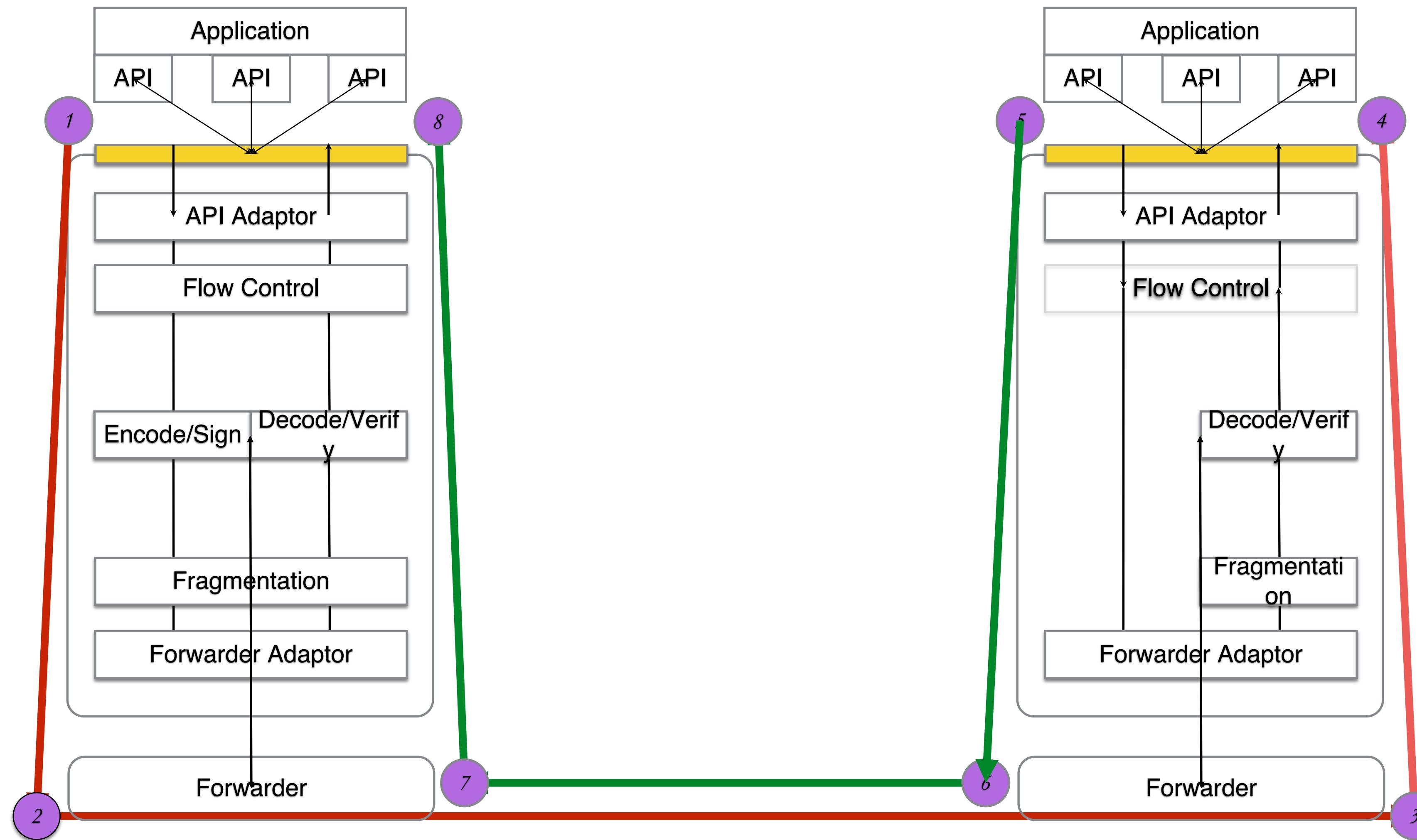
The CCN software architecture



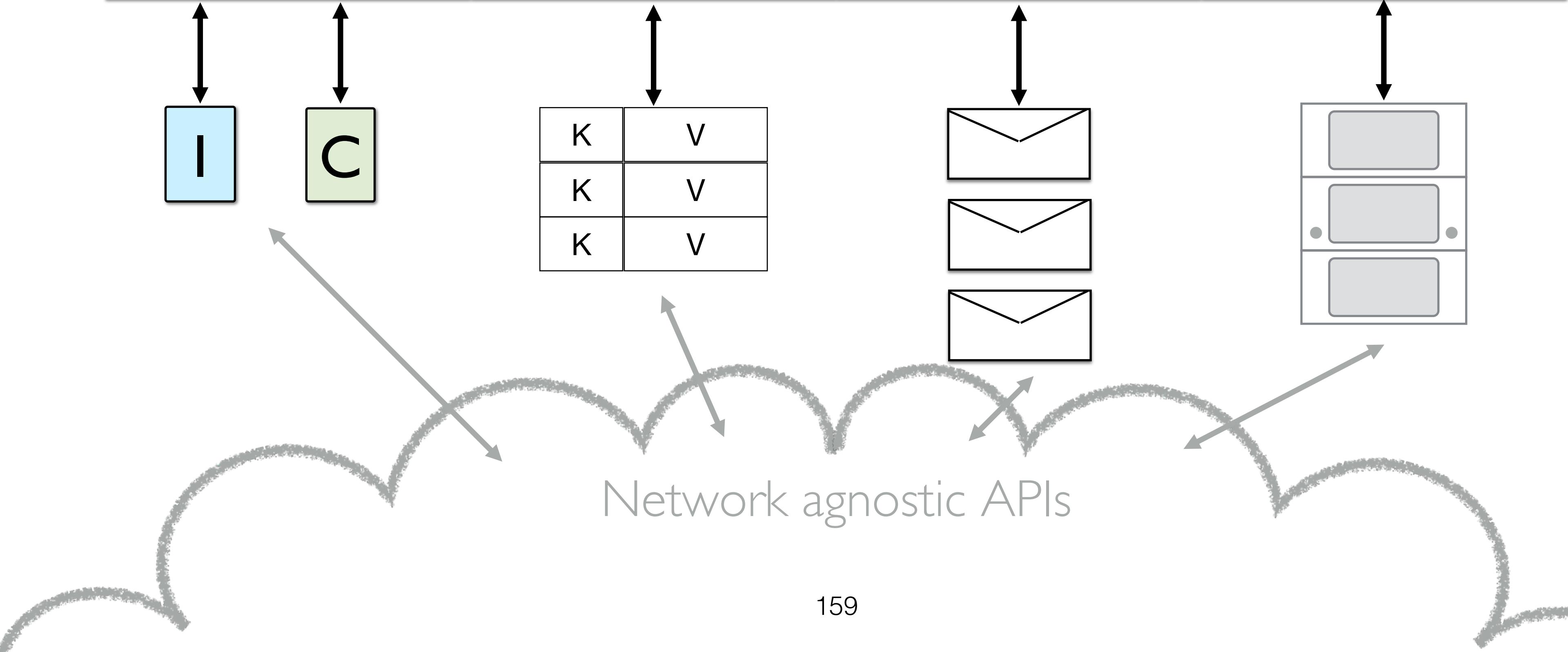
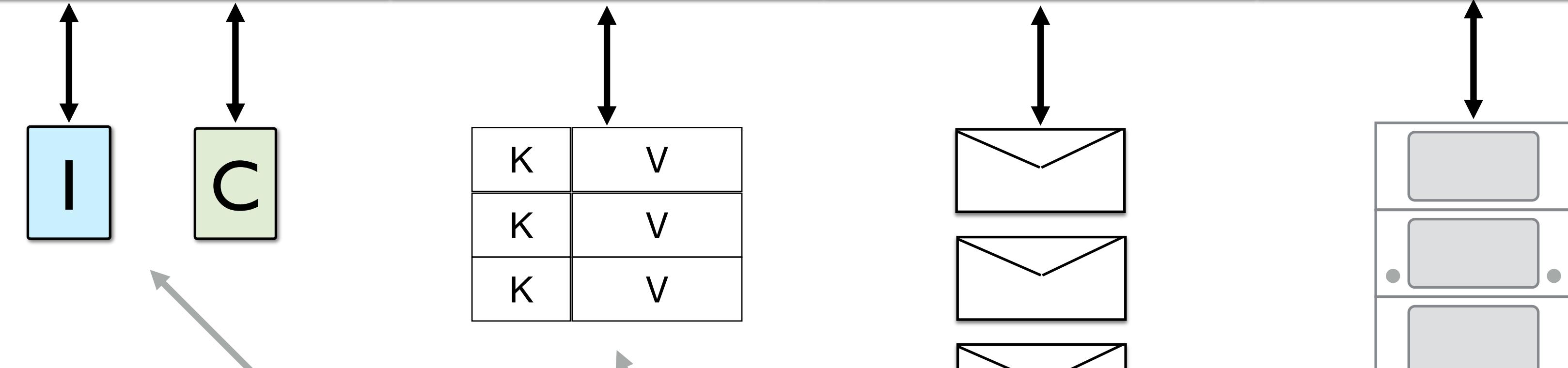
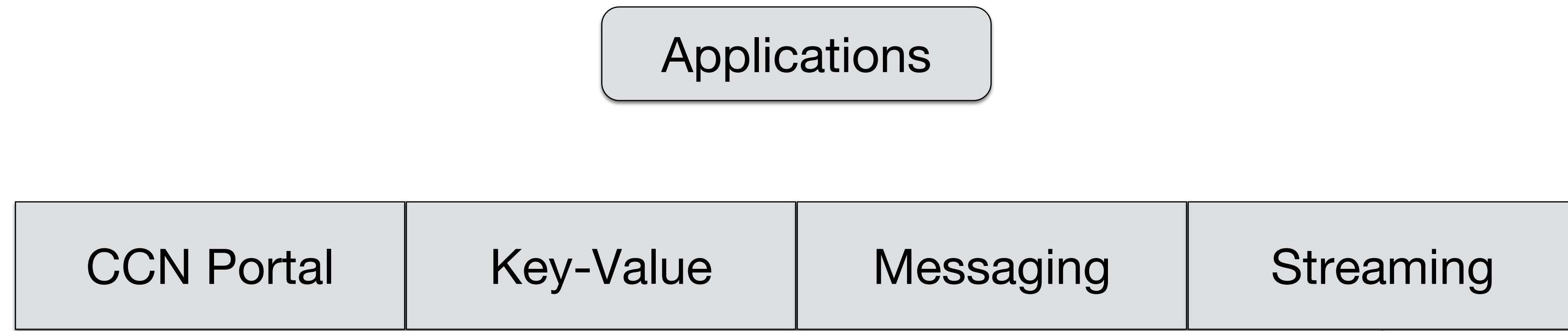
The CCN transport stack



Transport Stack



CCN Services and API characteristics



CCN - 1.0 Software

CCN 1.0 Software

What Is It?

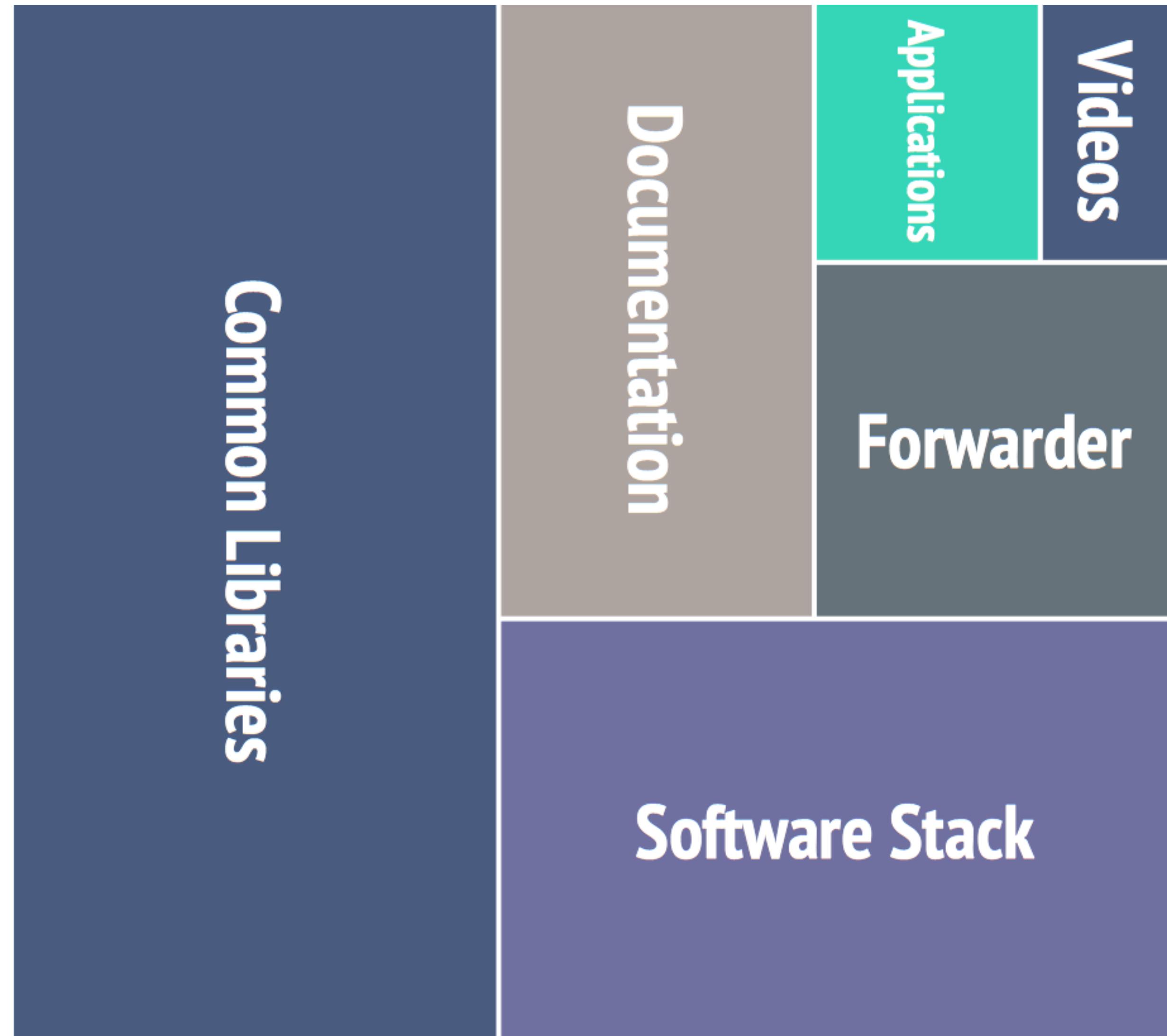
What Is Different?

How Do I Use It?

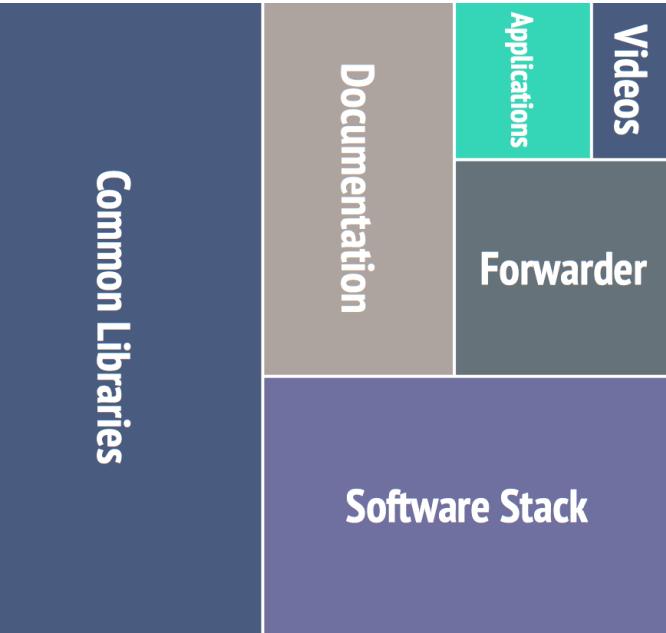
CCN 1.0 Software

What Is It?

- Application Programs
- Software Stack
- Common Libraries
- Documentation
- Instructional Videos



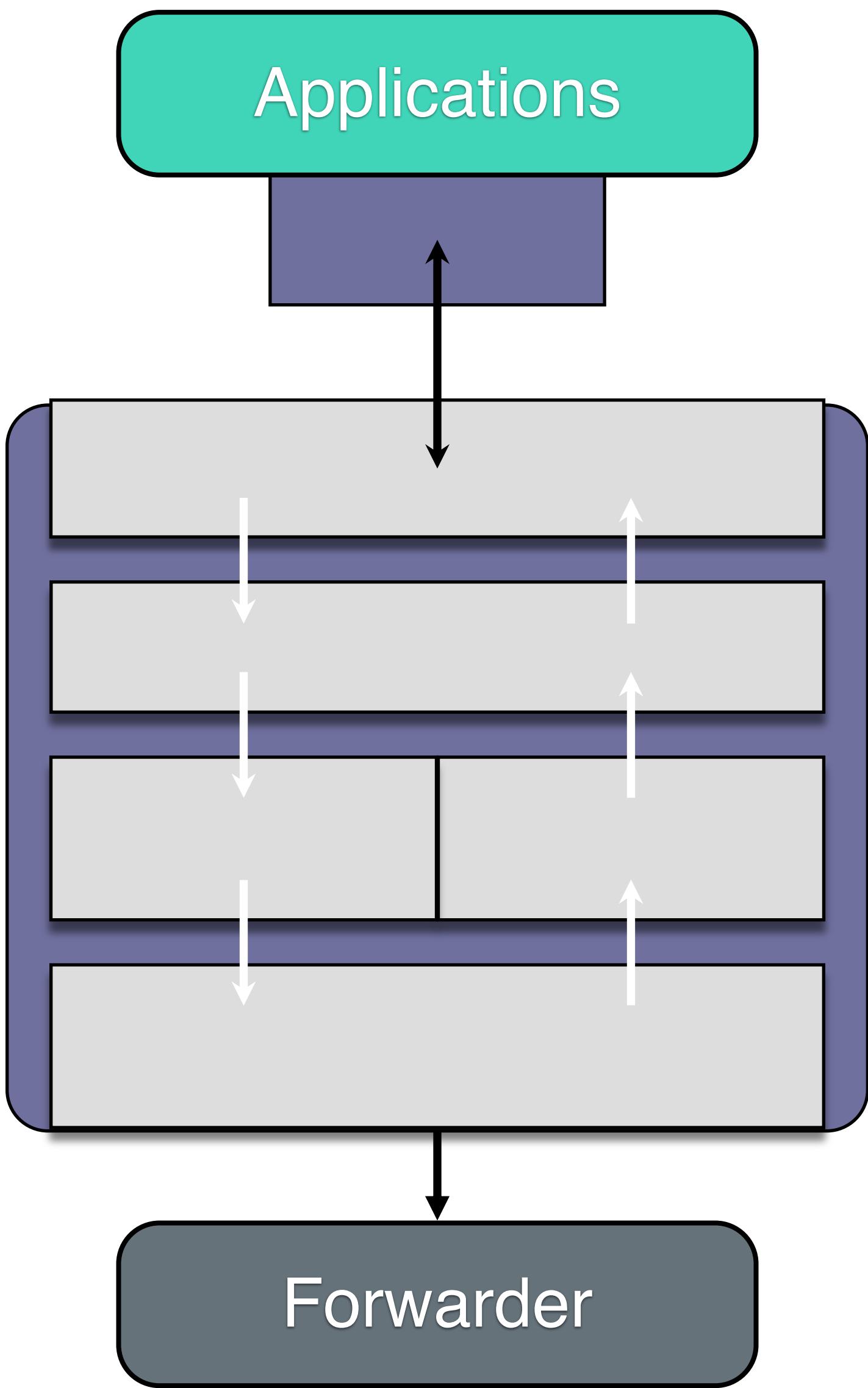
CCN 1.0 Software



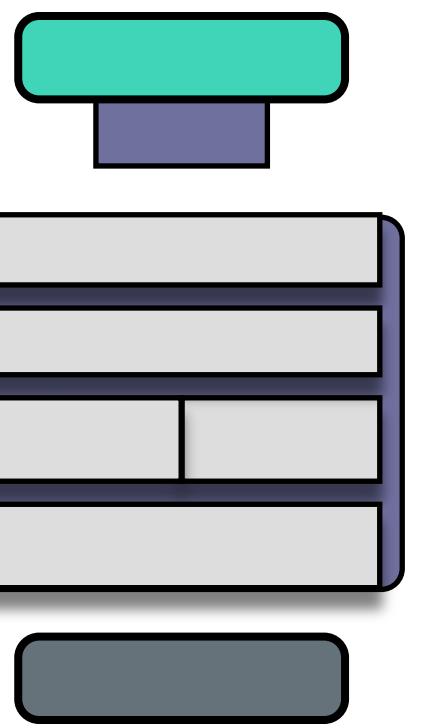
CCN Software Stack

Portal API

Transport Framework



CCN 1.0 Software



CCN Portal API

Streaming

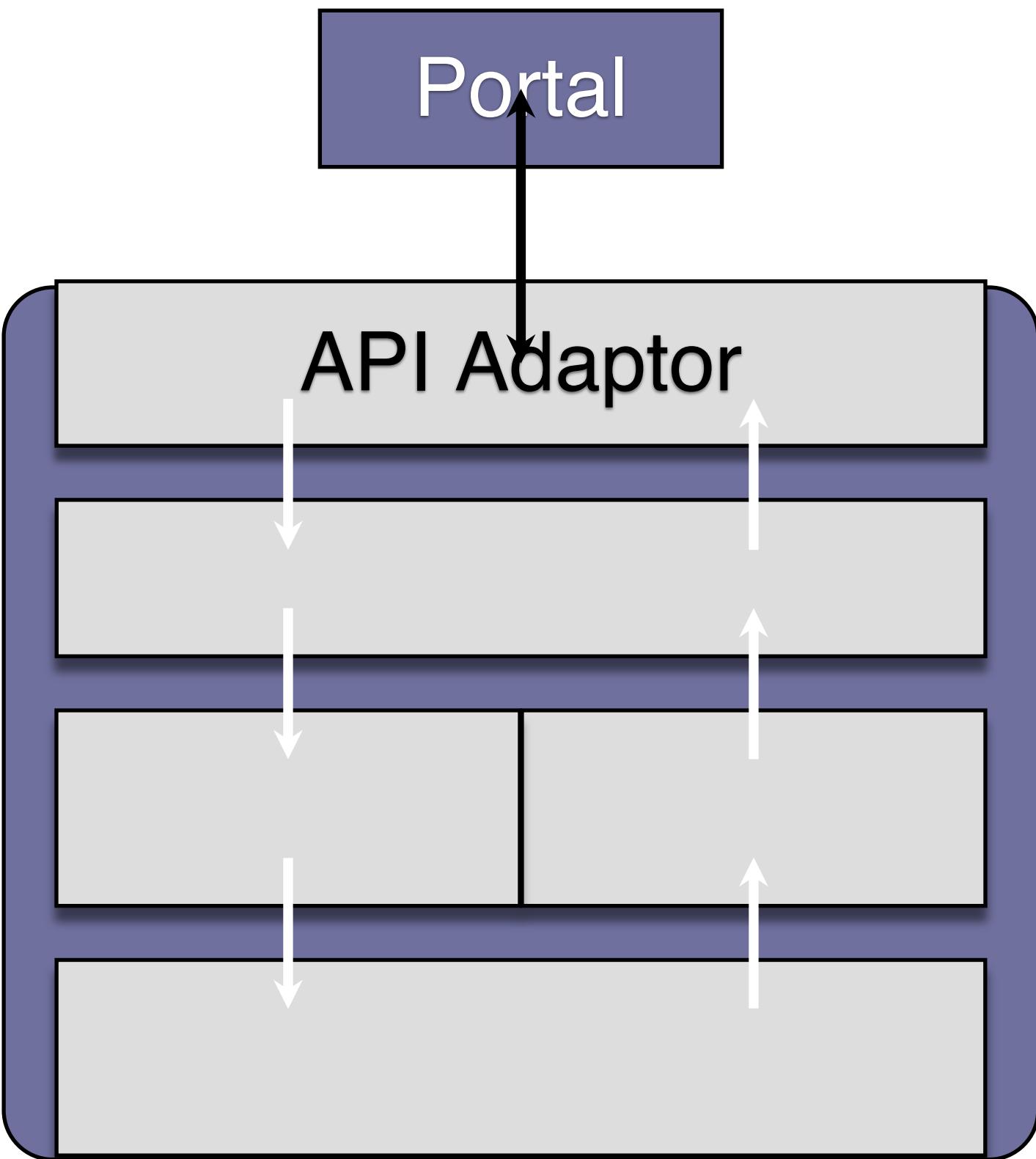
Datagram

Blocking

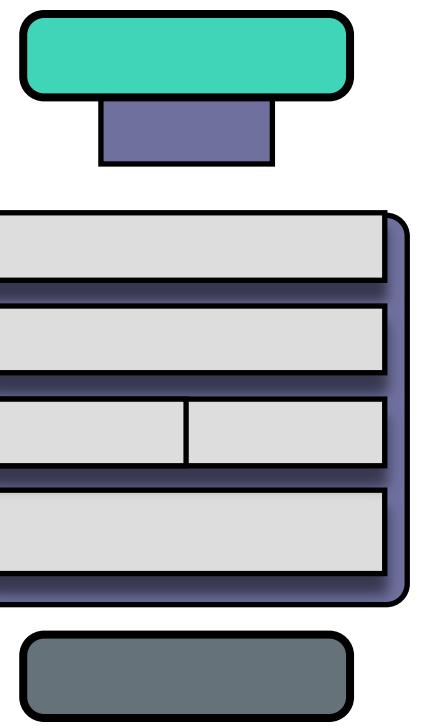
Non-blocking

Portal

`ccnxPortal_Listen`
`ccnxPortal_Read`
`ccnxPortal_Write`
`ccnxPortal_Release`
`ccnxPortal_Acquire`
`ccnxPortal_Ignore`
`ccnxPortal_IsEOF`
`ccnxPortal_IsError`



CCN 1.0 Software

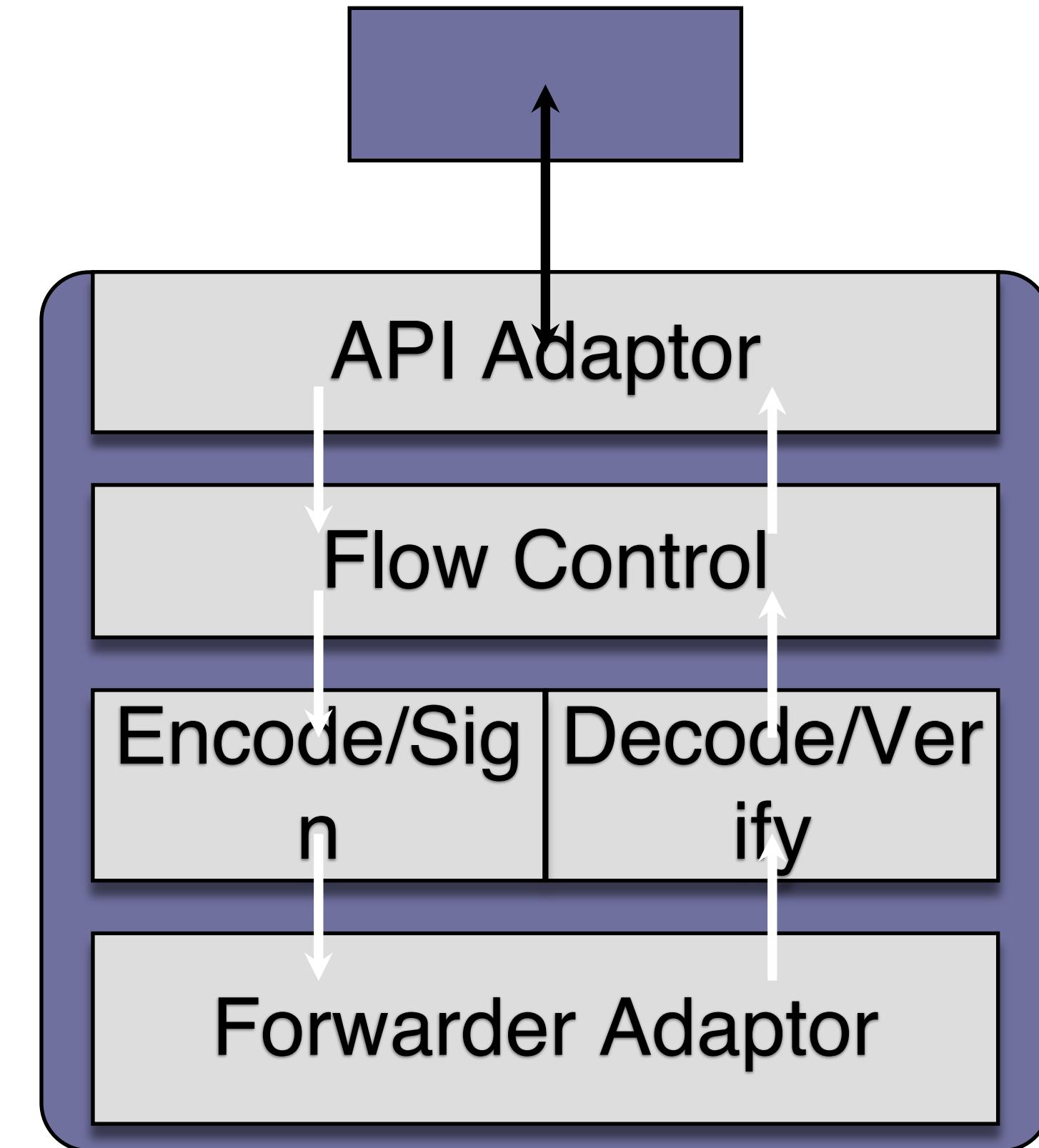


Transport Framework

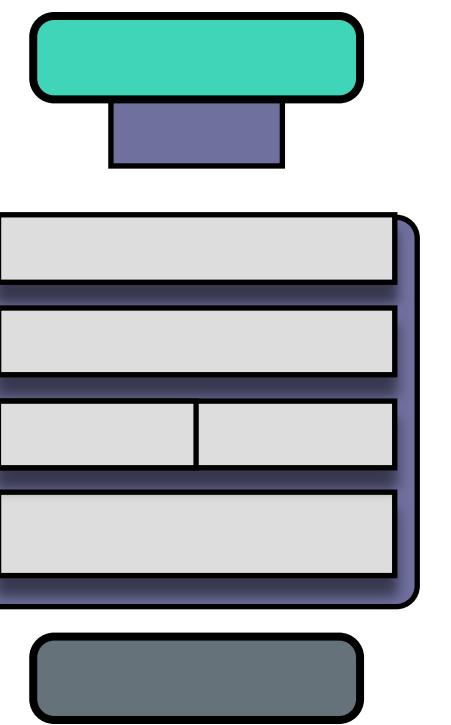
Component Based Design

Dynamically Plumbed

Dynamically Loaded Modules
(future)



CCN 1.0 Software



Easy to Use

- Simple application

- No special privileges

Configurable Cache Size

- In memory cache, 0 to ...

CCN 1.0 Packet Format

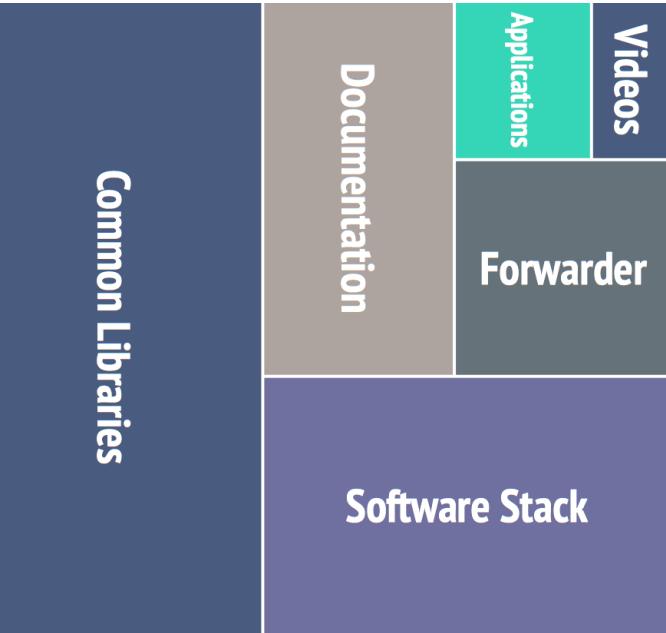
- TCP/IP encapsulation

- Native ethernet

Forwarder

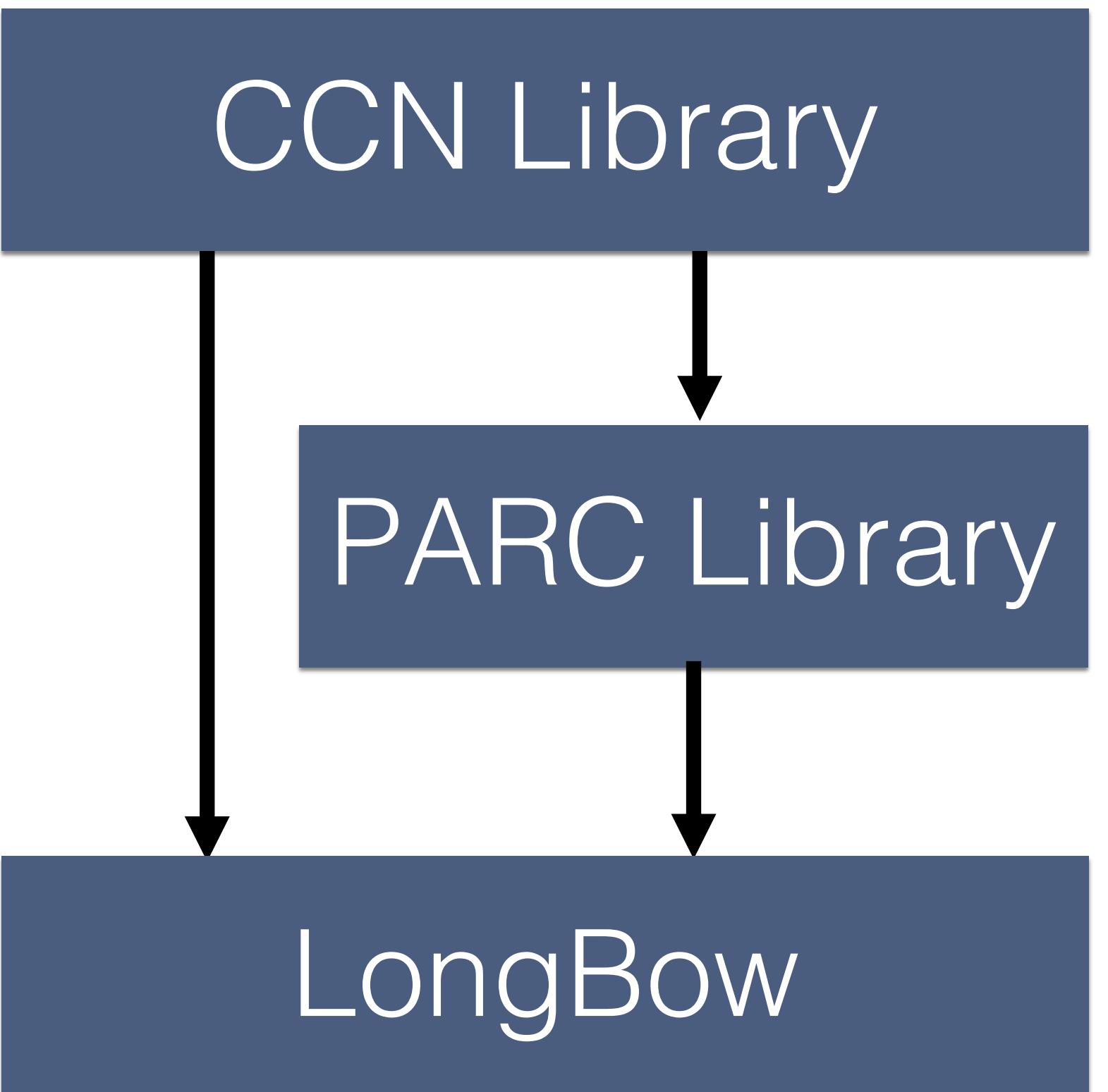
Built For Experimenting

CCN 1.0 Software

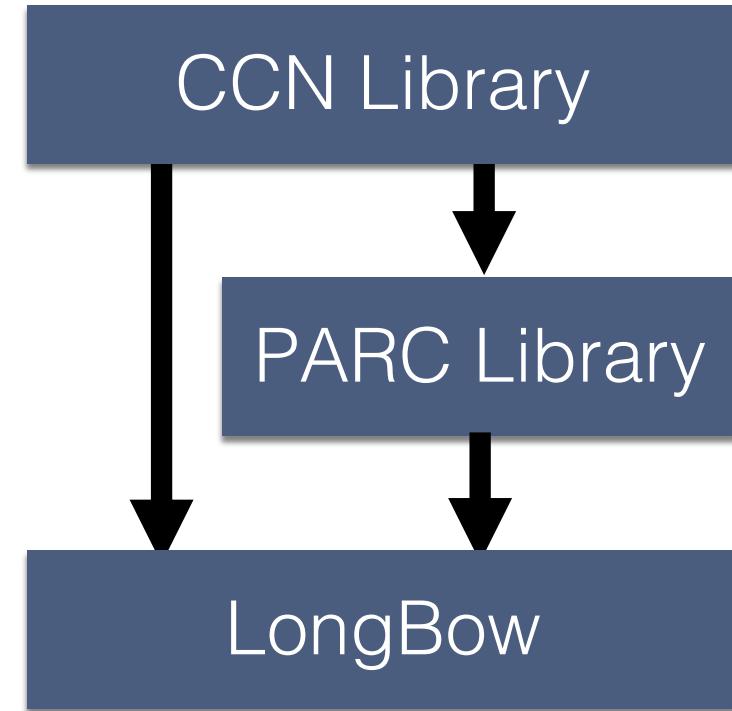


Common Libraries

Used By Everything
Reference Down,
No Reference Up



CCN 1.0 Software



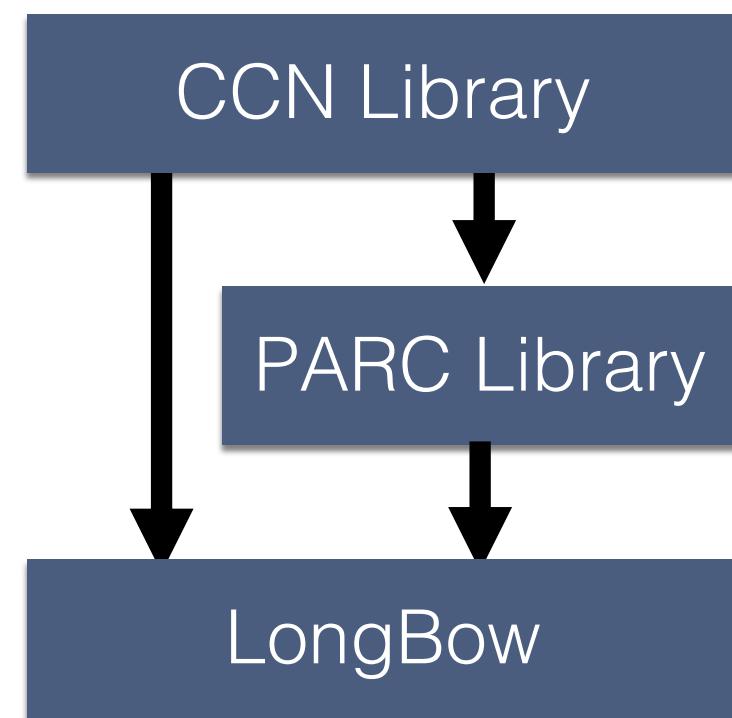
CCN Library

Application Writer's View

CCNxContentObject
CCNxName
CCNxInterest
CCNxPortalMessage

CCNxNameSegmentType
CCNxKeystoreUtilities

CCN 1.0 Software

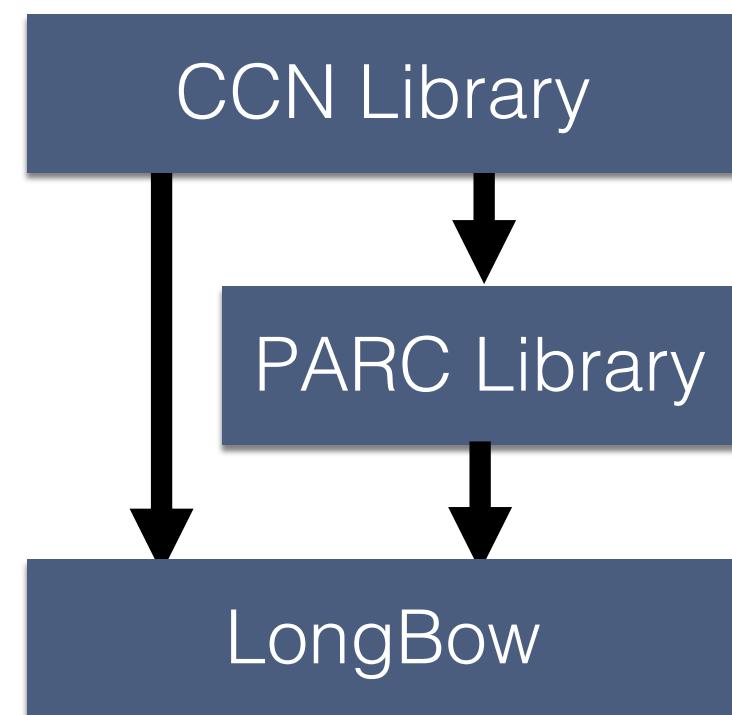


CCN Library

API Writer's View

CCNxControl
CCNxMessage
CCNxName
CCNxInterest
CCNxContentObject
CCNxKeyLocator

CCN 1.0 Software



CCN Library

Component Writer's View

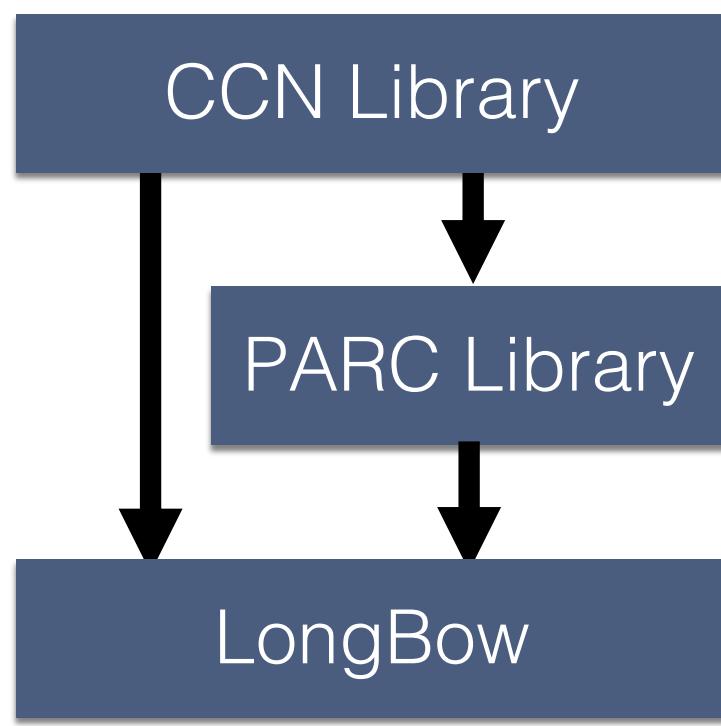
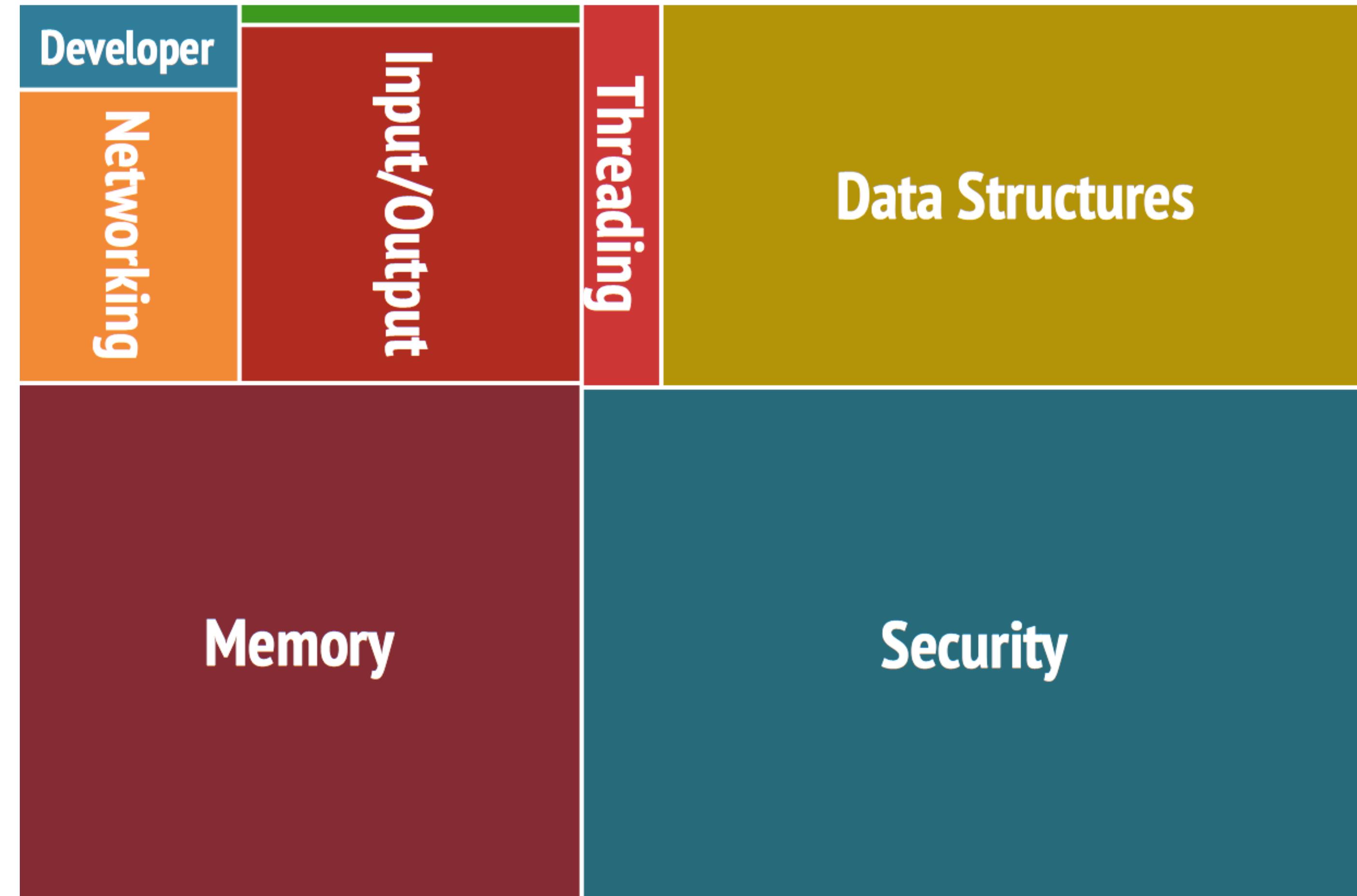
CCNxValidationFacade
CCNxContentObject
CCNxControlFacade
CCNxTlv^{CCNxControl}_{CCNxName}
CCNxTlvDictionary
CCNxMessage
CCNxWireFormatFacade
CCNxJson
CCNxNetworkBuffer

CCN 1.0 Software

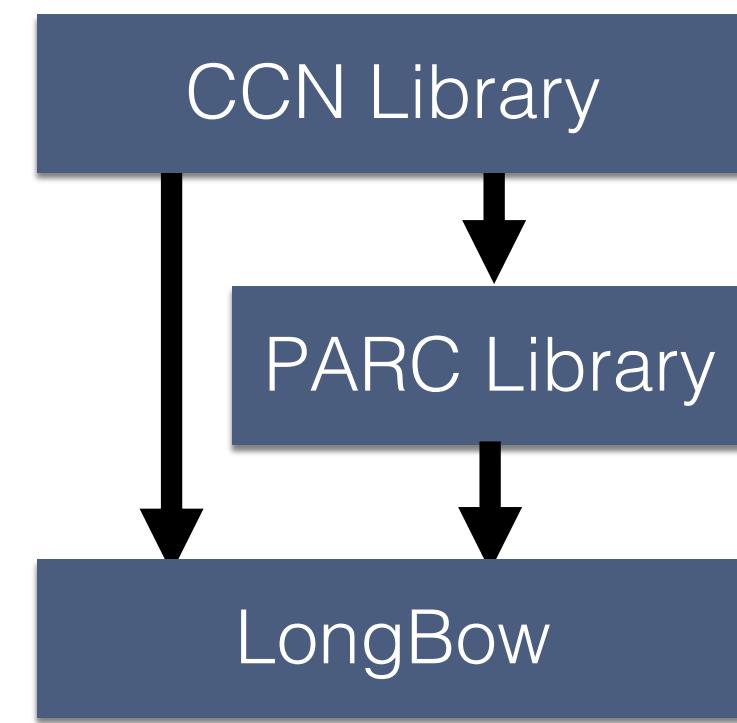
PARC Library

C Utility Functions

General Purpose



CCN 1.0 Software



Runtime Assertions and Traps

assertTrue	trapIllegalValue
assertFalse	trapNotImplemented
assertNull	TrapOutOfBounds
assertNotNull	trapOutOfMemory
	trapUnexpectedState

LongBow

Write Better C Programs

Native C Unit Test Framework

- xUnit-style testing for C (in C)
- Integrates with runtime assertions and traps
- Integrated with automake, Xcode, Eclipse

CCN 1.0 Software

What Is Different?

Written for Extension and Experimentation

Written for People

Measured

CCN 1.0 Software

Written for Experimentation

Interface Based Architecture

Clear Separation of Concerns

Simple to Substitute Different Implementations

Modular Design

Promotes Extensibility

Interoperability Testing

Graduated/Progressive Implementation

CCN 1.0 Software

Written For People

Human Factors Emphasis

Documentation

Function Documentation

Tutorial Guides

Consistent Design and Style

Measured Software

CCN 1.0 Software

Documentation

100% Coverage

Module, Function, Enumeration Type

IDE Integration

Printed and Online

// Get the actual contents of the specified chunk of the file.

```
PARCBuffer *payload = tutorialFileIO_GetFileChunk(fullFilePath
```

Declaration `PARCBuffer *tutorialFileIO_GetFileChunk(const char *fileName, size_t chunkSize, uint64_t chunkNumber)`

Description Given a `fileName` and `chunk number`, retrieve that chunk from the specified file. The contents of the chunk are returned in a `PARCBuffer` that must eventually be released via a call to `parcBuffer_Release(&buf)`. The `chunkNumber` is 0-based.

Parameters `fileName[in]` A pointer to a string containing the name of the file to read from.

`chunkSize[in]` The maximum number of bytes to be returned in each chunk.

`chunkNumber[in]` The 0-based number of chunk to return from the file.

Returns A newly created `PARCBuffer` containing the contents of the specified chunk.

Declared In [tutorial_FileIO.h](#)

CCN 1.0 Software

Consistent Design and Style

Uniform Code Style

Clean and Clear Naming

High Cohesion

Low Coupling

The figure displays four screenshots of PARC design documents, each from the "Computing Science Laboratory" at the "Palo Alto Research Center".

- PARC C Style Guide**: A document by Glenn Scott. It includes an abstract, keywords, and a table of contents.
- PARC Library Canonical C Function Name Conventions**: A document by Glenn Scott. It includes an abstract, keywords, and a table of contents.
- PARC Design Namespaces for C Programs**: A document by Glenn Scott. It includes an abstract, keywords, and a table of contents.
- PARC Design Managing Modularity and Coupling in C Programs**: A document by Glenn Scott. It includes an abstract, keywords, and a table of contents.

The screenshots show the layout of the documents, including the header information, the main content sections, and the table of contents at the bottom of each page.

CCN 1.0 Software

Measured

Style Conformance

Naming Conformance

Unit Testing

Documentation Coverage

Complexity Management

Developer Tools

Readiness and Acceptance

CCNx Dashboard

Source Code

	Main	Testing	Test Coverage
Forwarder	19	16	<div style="width: 74%; background-color: #007bff; height: 10px;"></div>
APIs	14	8	<div style="width: 57%; background-color: #007bff; height: 10px;"></div>
Transport	27	15	<div style="width: 55%; background-color: #007bff; height: 10px;"></div>
CCN Library	25	14	<div style="width: 56%; background-color: #007bff; height: 10px;"></div>
PARC Library	38	21	<div style="width: 55%; background-color: #007bff; height: 10px;"></div>
LongBow	10	...	<div style="width: 100%; background-color: #007bff; height: 10px;"></div>

Total Lines

136k

77k

74%

Developer Documentation

	Print	HTML	Coverage
Forwarder	315	354	<div style="width: 88%; background-color: #007bff; height: 10px;"></div>
APIs	301	239	<div style="width: 79%; background-color: #007bff; height: 10px;"></div>
Transport	438	471	<div style="width: 92%; background-color: #007bff; height: 10px;"></div>
CCN Library	604	234	<div style="width: 39%; background-color: #007bff; height: 10px;"></div>
PARC Library	907	413	<div style="width: 45%; background-color: #007bff; height: 10px;"></div>
LongBow	225	175	<div style="width: 77%; background-color: #007bff; height: 10px;"></div>

Total Pages

2,864

1,972

92%

Code Complexity

Cyclomatic

Forwarder

1.77

APIs

1.88

Transport

2.43

CCN Library

2.29

PARC Library

1.97

LongBow

2.00

Vocabulary

55

51

70

45

43

39

Modularity

Forwarder



APIs



Transport



CCN Library



PARC Library



LongBow



Average

2.00

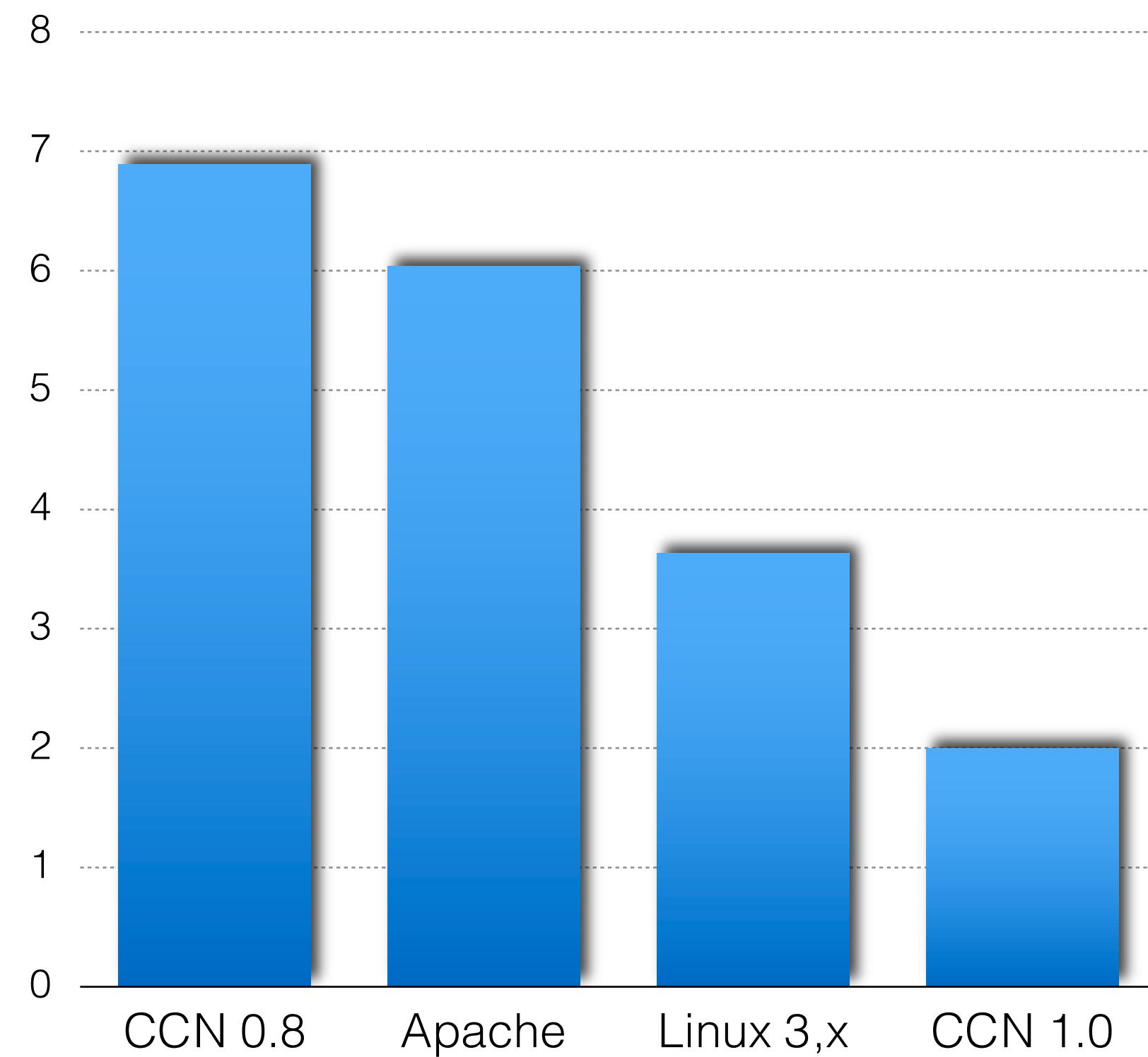
50

Average

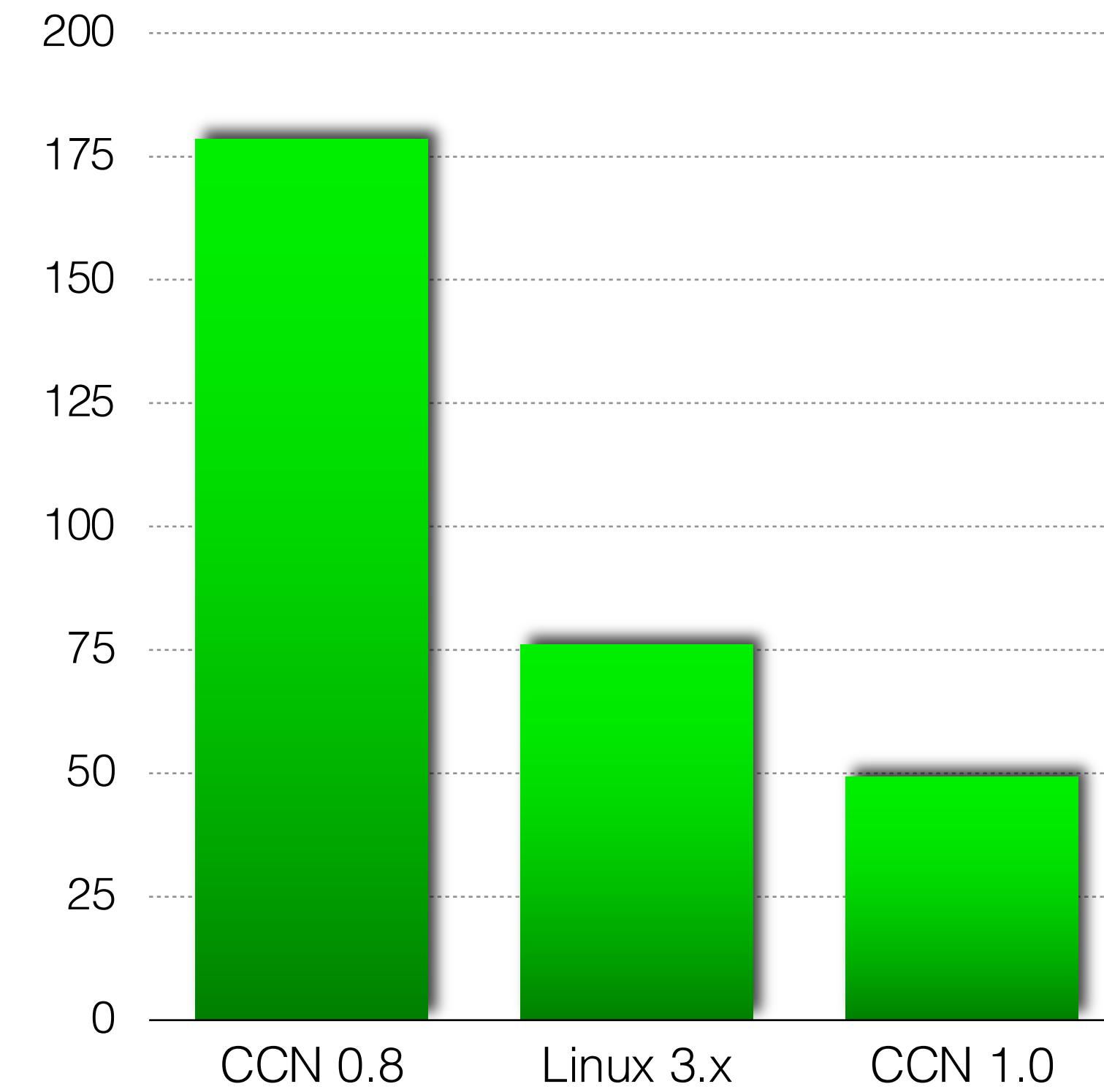
63%

CCN 1.0 Software

Cyclomatic Complexity



Vocabulary



CCN 1.0 Software

How Do I Use It?

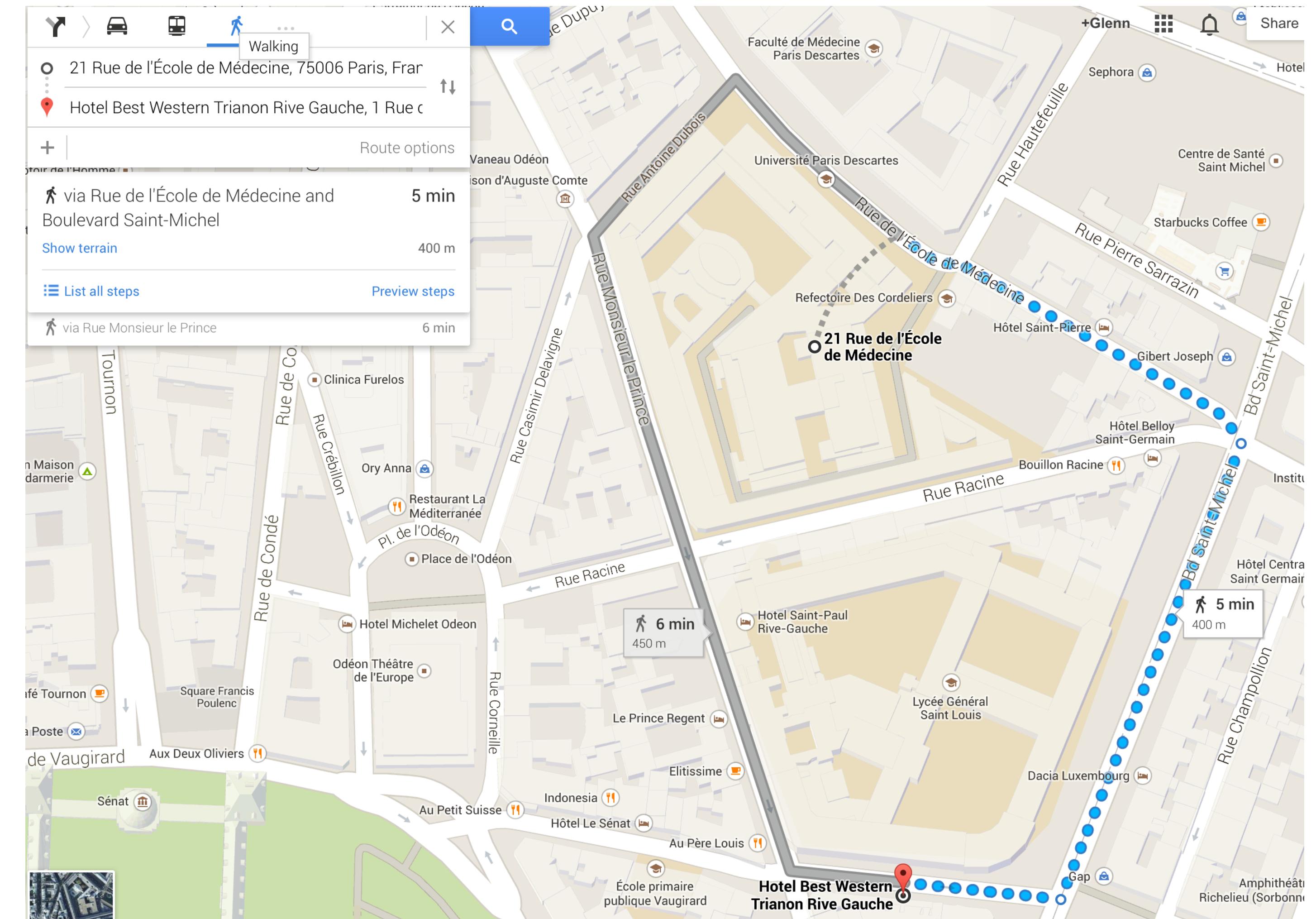
Install CCN 1.0 Software

Use Portal API

Link Application with Libraries

Run

Sit back, enjoy





parc[®]

A Xerox Company

Thank you

<http://www.ccnx.org/>

CCN - Extra Slides

CCN - Examples

Sample use case

Web page

A simplified example

Web page

Step 1 - Request web page manifest

Retrieve the manifest of the web page

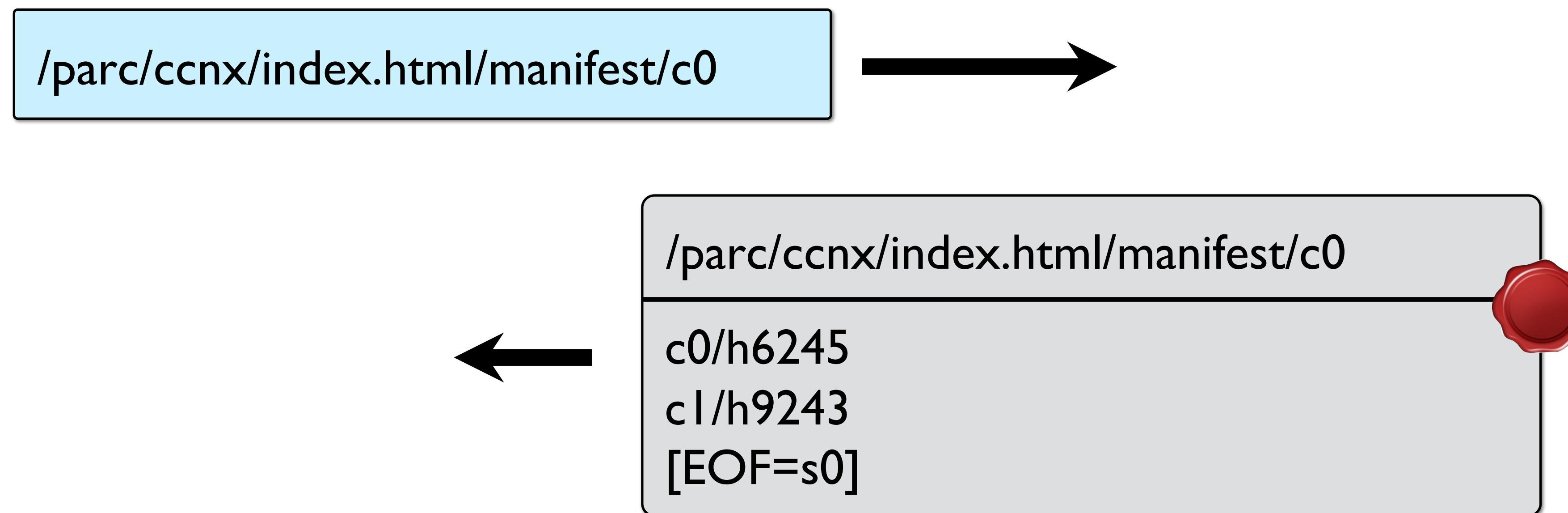
Step 2 - Request web page content

Download the web page content (html)

Step 3 - Request references

Download embedded references (images, css, etc)

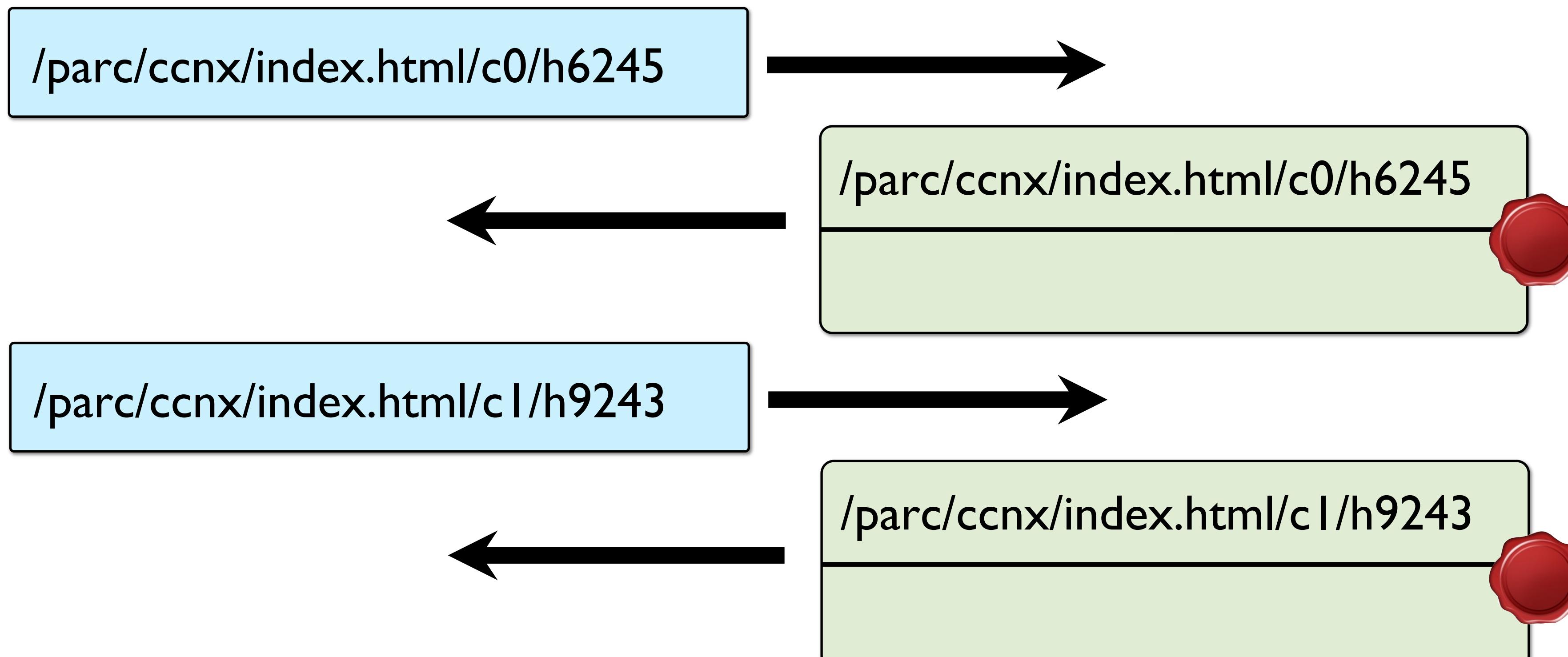
Request page manifest



Request the list of chunks for the main webpage

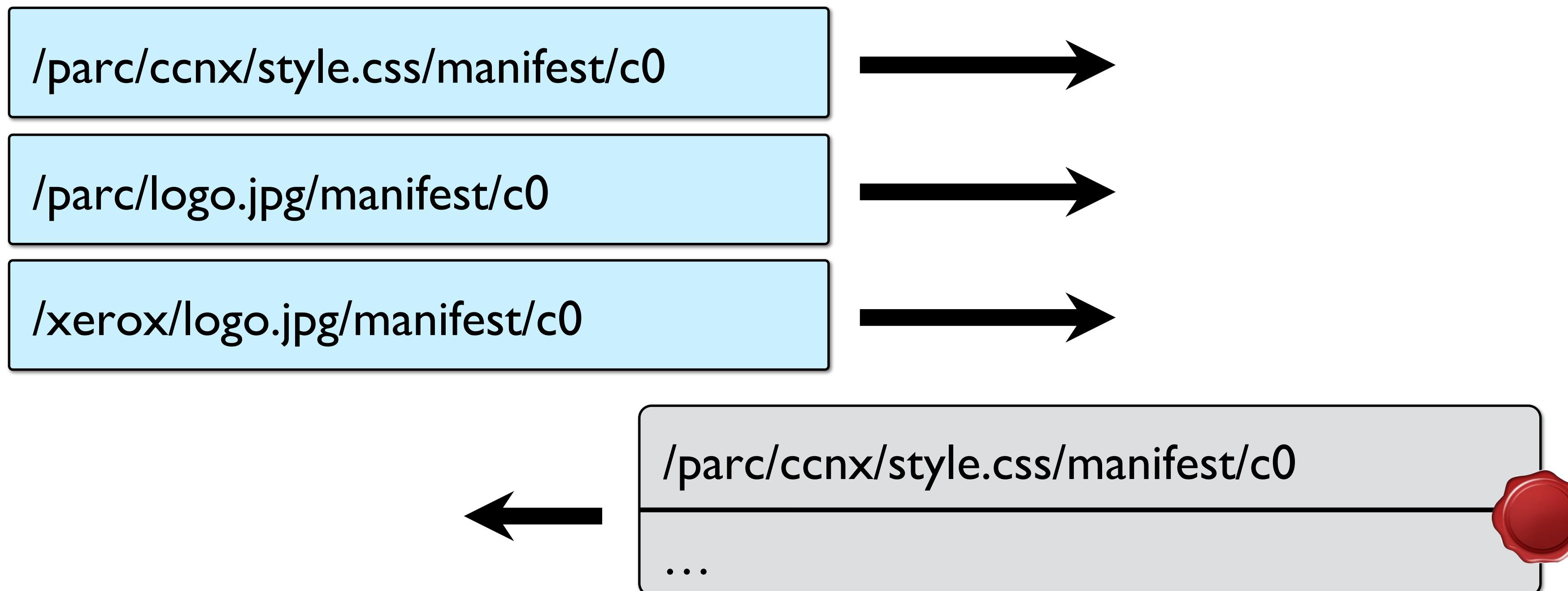
Each chunk identified with a hash

Request page content



Request each chunk using name and hash

Retrieve references



Sample use case

Netflix

A simplified example

Netflix

Step 1 - Login

Contact Netflix to login, retrieve session key

Step 2 - Request movie network name

Get the actual network name

Step 3 - Request personal movie key

Get a key to decrypt the movie



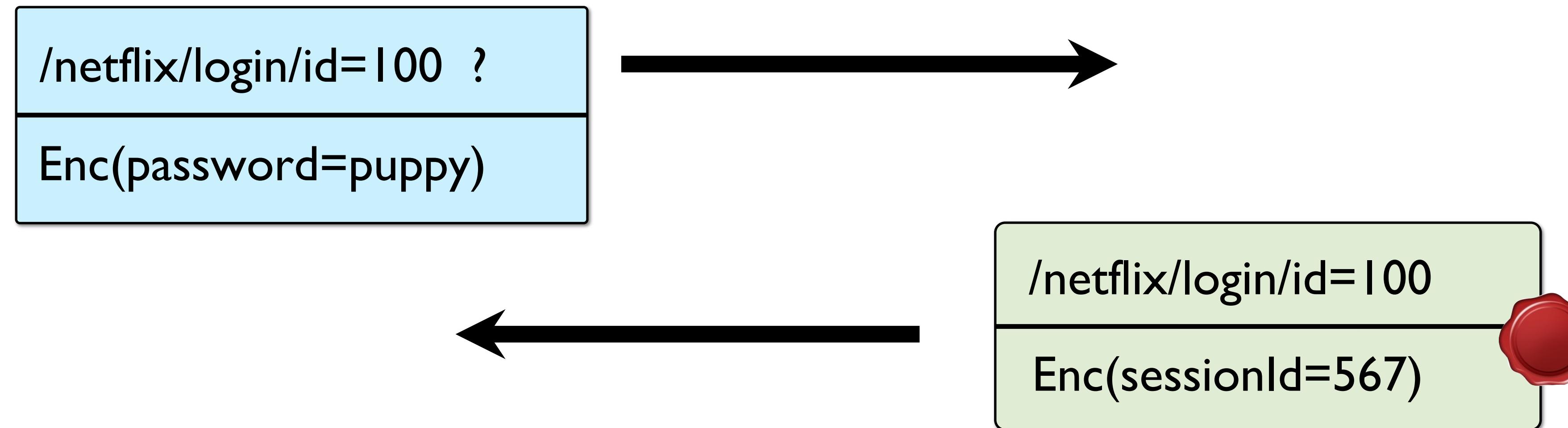
Step 4 - Request movie manifest

Get the movie manifest object

Step 5 - Request movie stream

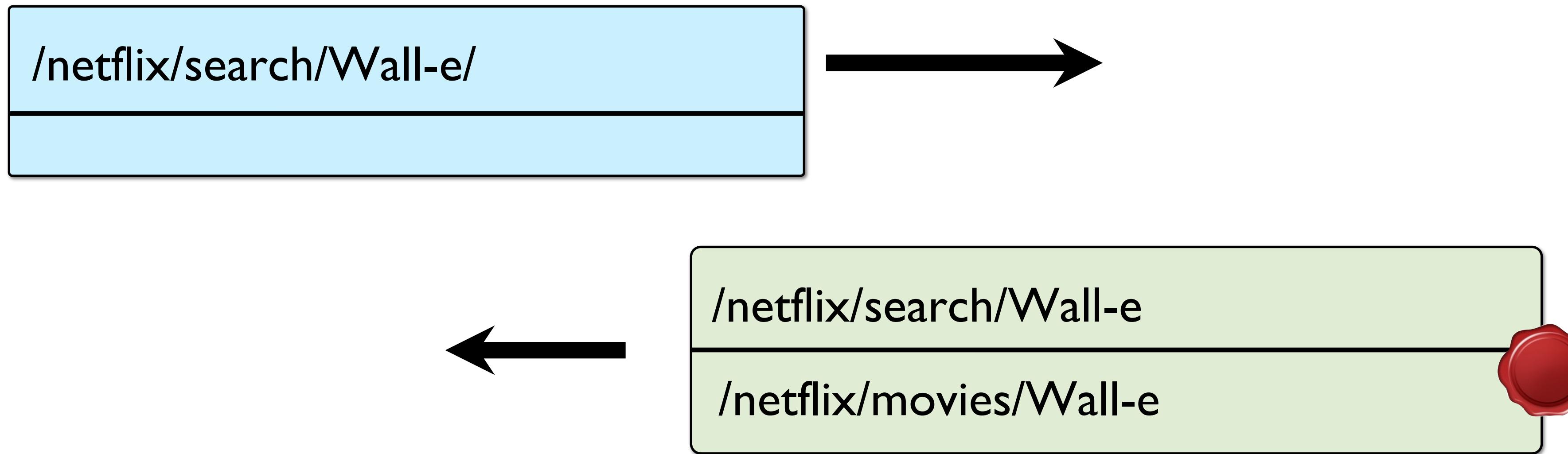
Download the actual movie contents based on the manifest.

Login to service



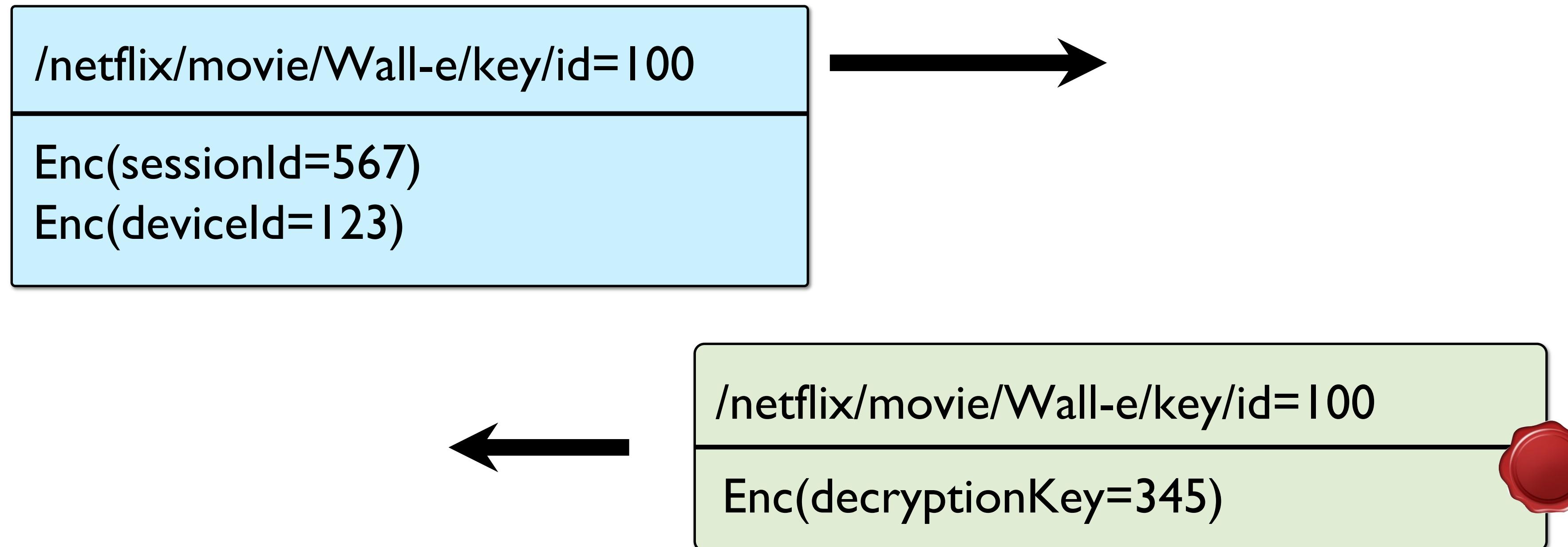
Request a session ID from Netflix using our login ID
Authenticate sending our password, encrypted

Get movie network name



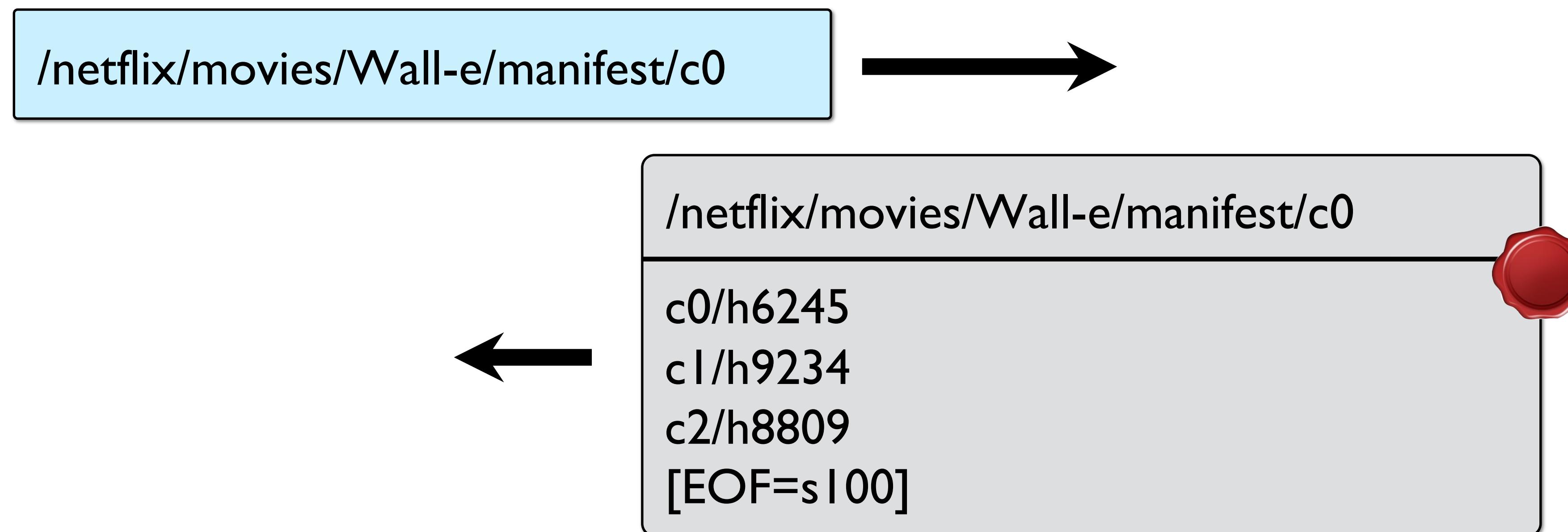
Search Netflix for the movie Wall-e
Get back a network name of the movie

Request movie key



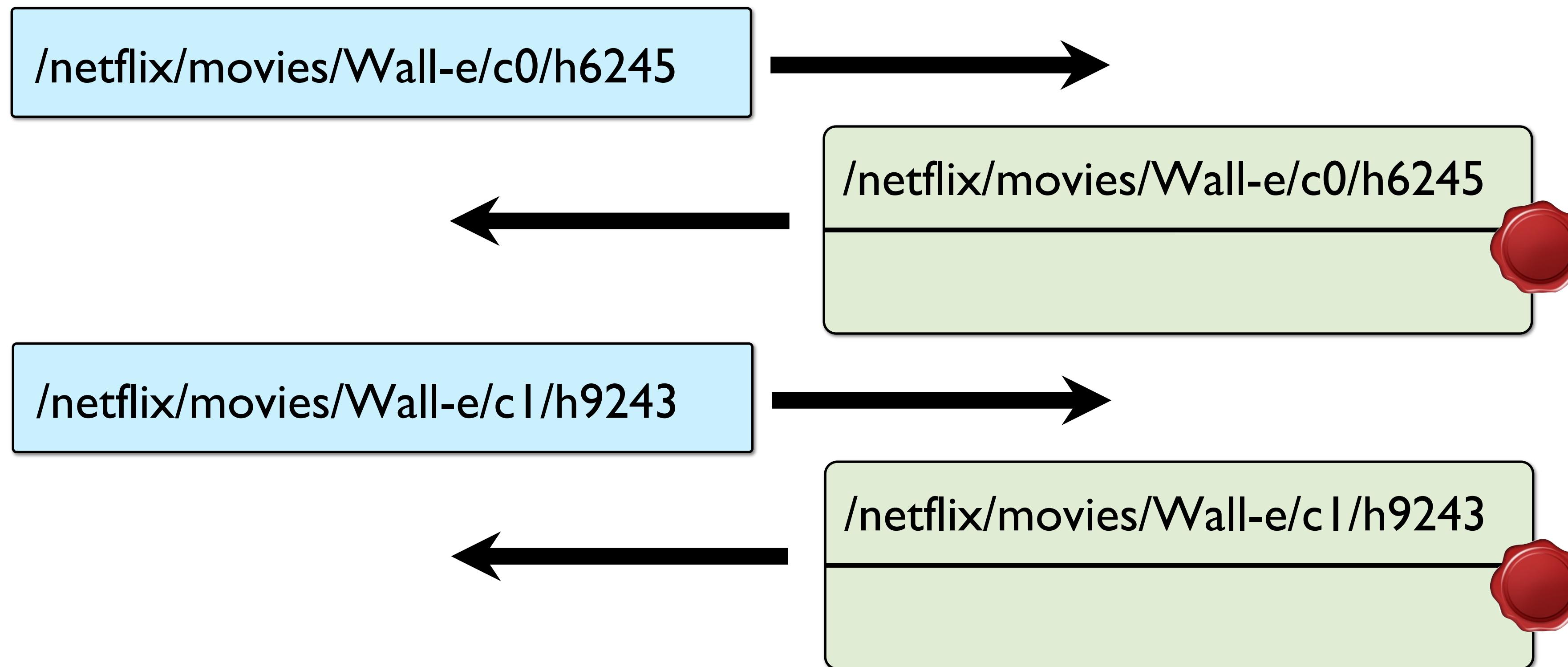
Request a personal key for the movie we want to watch
Send session ID and device ID

Request movie manifest

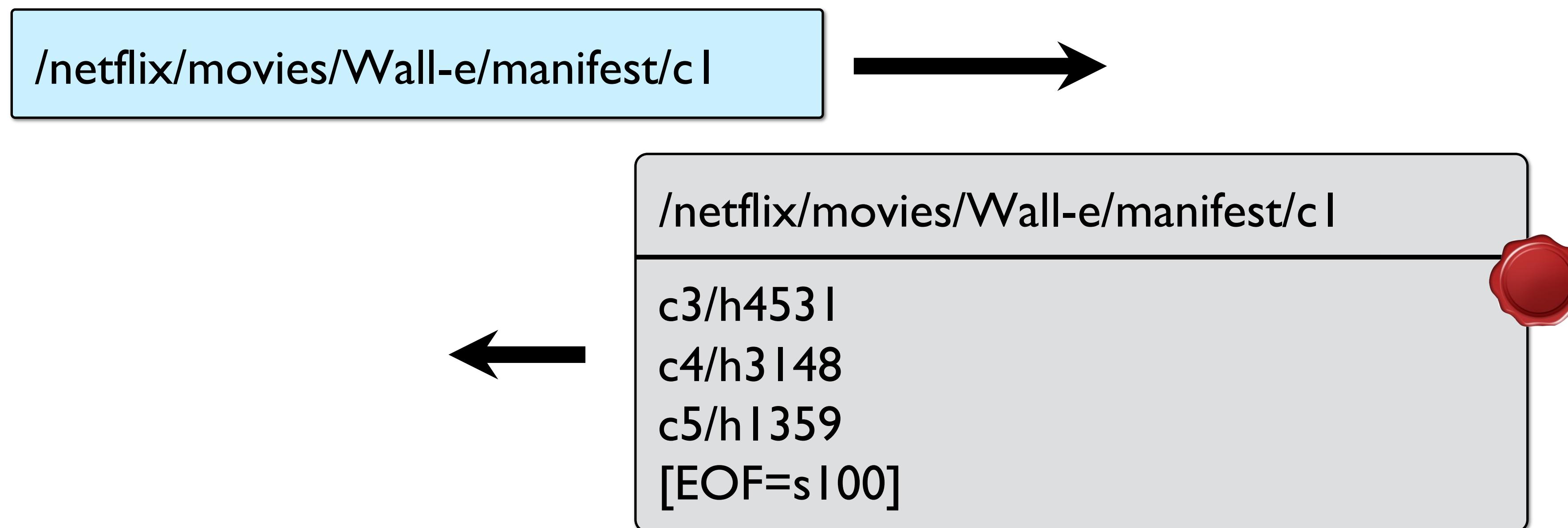


Request a list of segments for the movie
Each segment identified with a hash

Request movie content

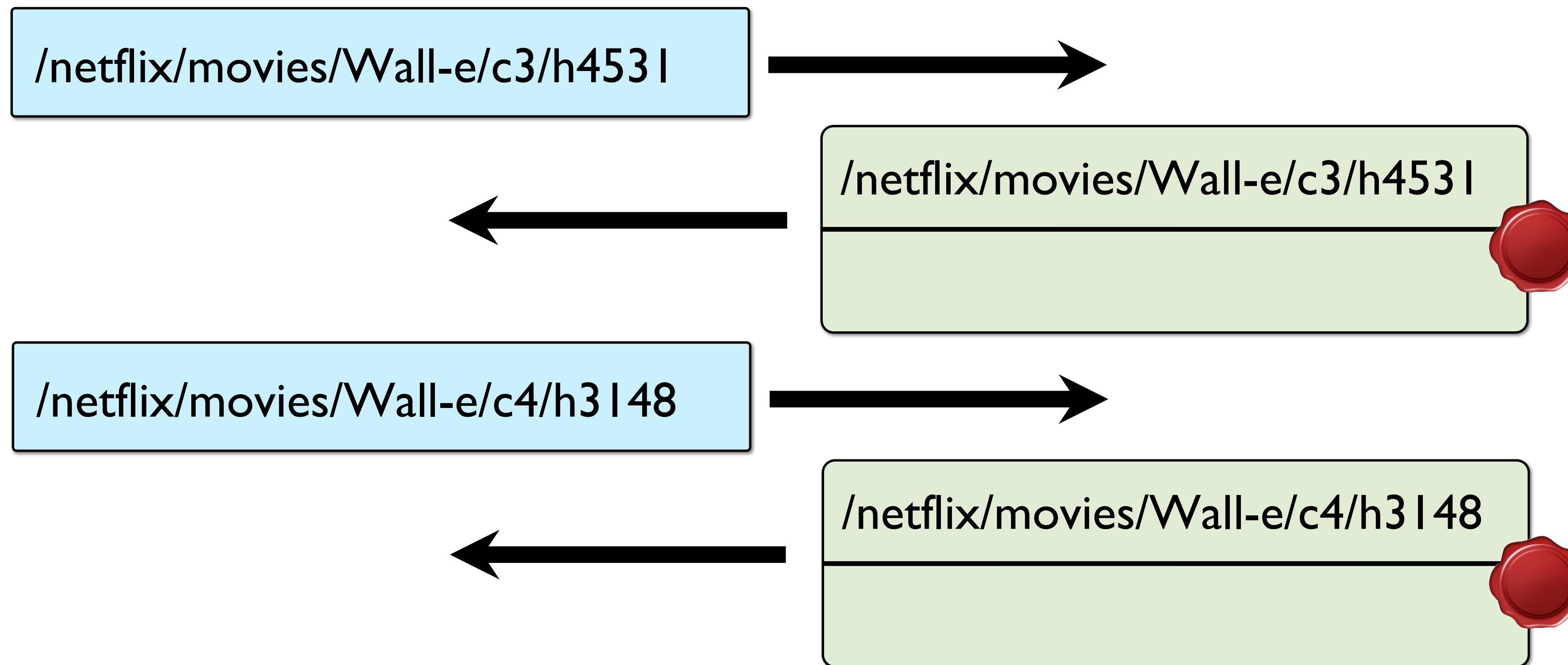


Request more manifest



Request a list of segments for the movie
Each segment identified with a hash

Request more movie content



Sample use case

Phone call

A simplified example

Phone Call

Step 1 - Setup call

Contact call agent at callee

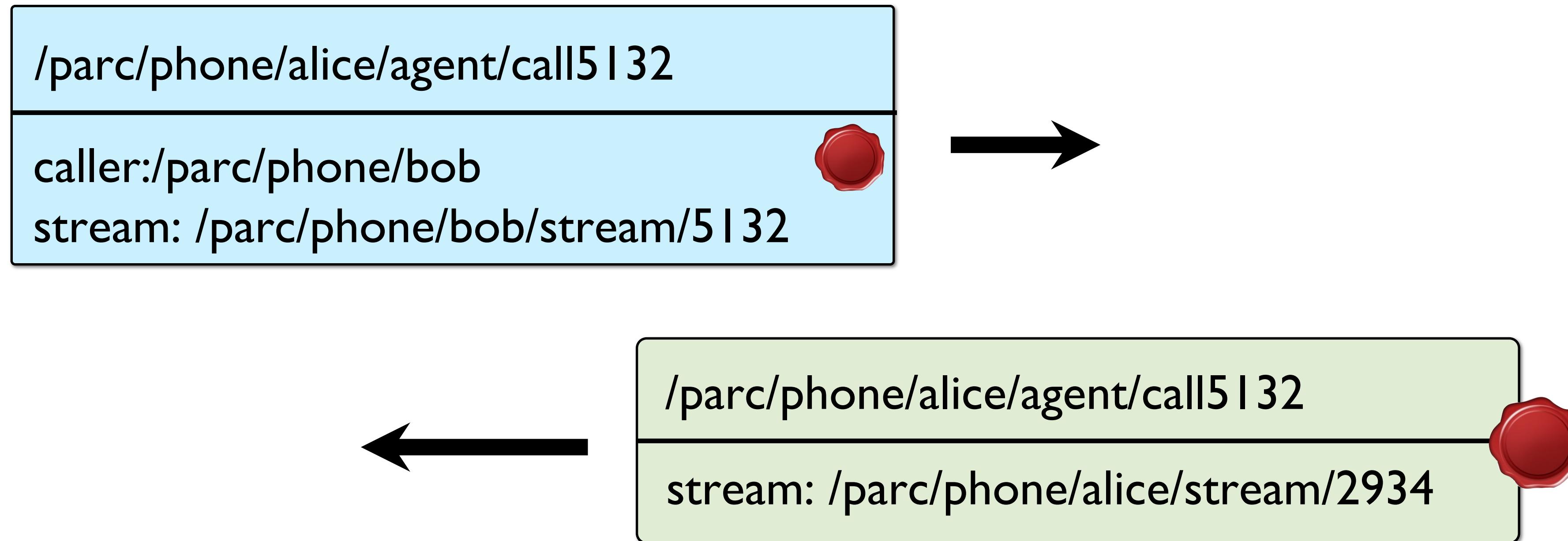
Step 2 - Request caller stream

Request audio stream from caller

Step 3 - Request callee stream

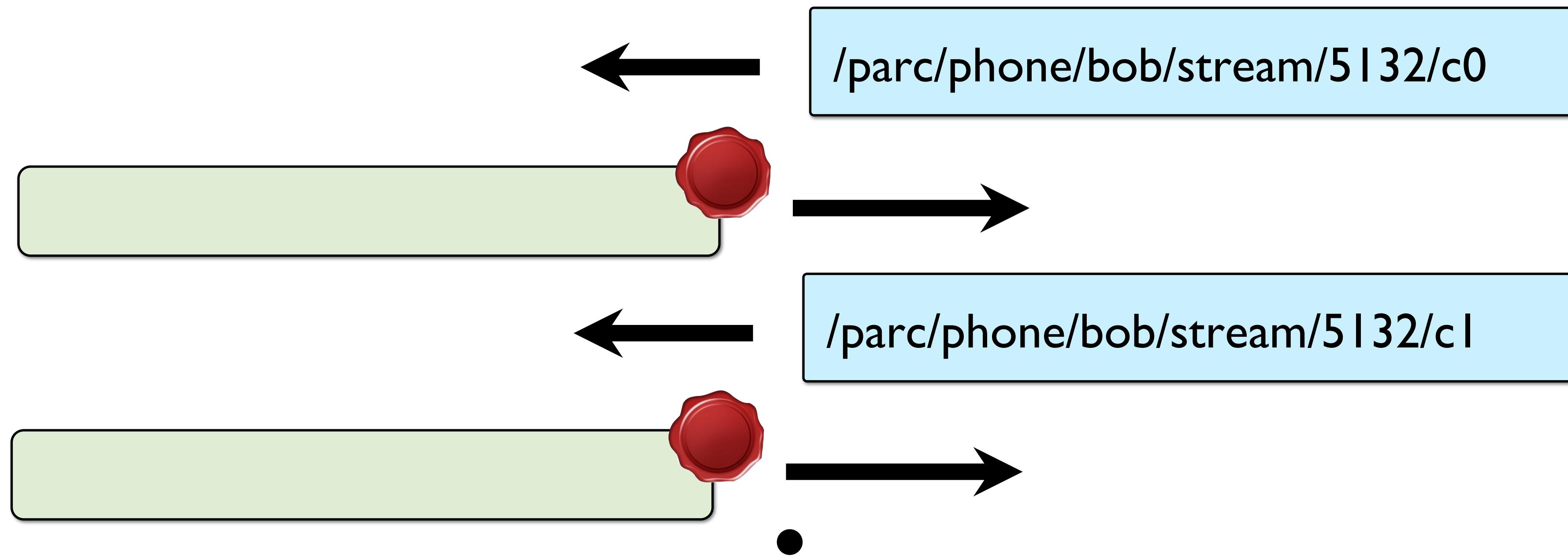
Request audio stream from callee

Setup call - Bob calls Alice



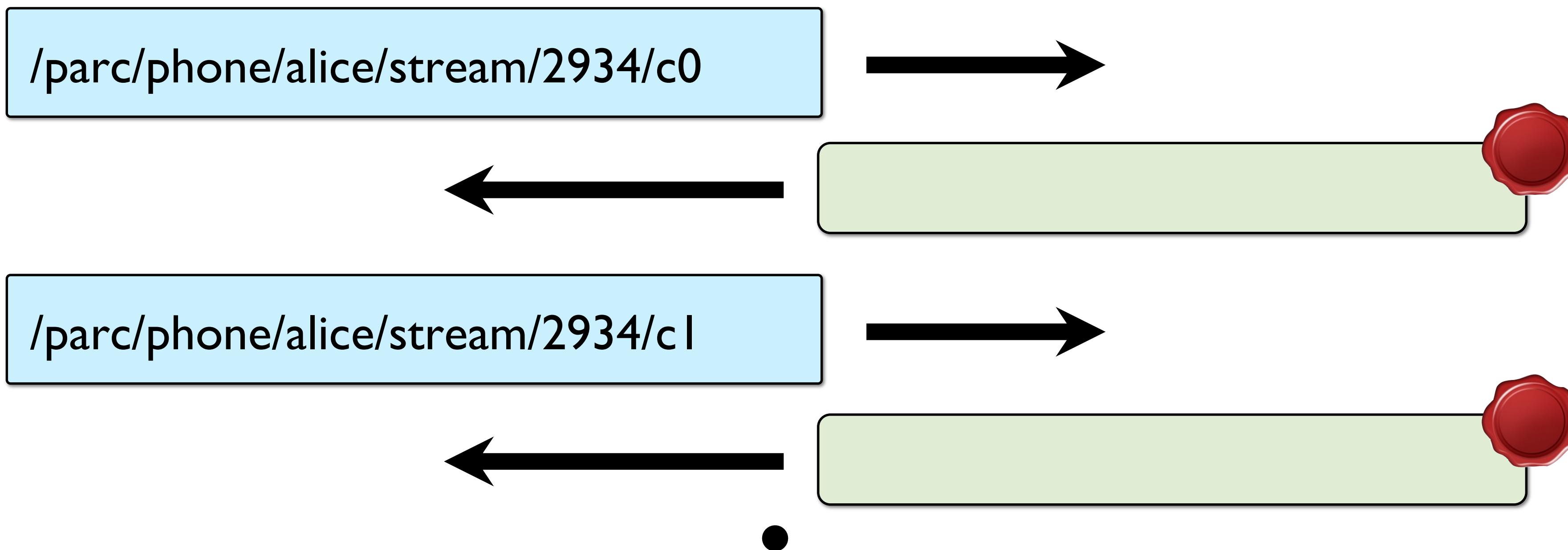
Setup a call from Bob to Alice. Bob tells Alice of the caller stream,
Alice tells bob of the callee stream.

Request caller stream



Alice requests Bob's audio stream

Request callee stream



Bob requests Alice's audio stream

Protocol Changes - Structural

Improved packet flexibility

Segmented packet - Using a static header, optional headers and message allows data to be carried efficiently at the right abstraction layer.

TLV packet format - Efficient and easy to parse packet structure with enough flexibility to extend and update the protocol.

Protocol Changes - Cleanup

Streamlined matching

Exact matching - Efficiently and unambiguously match an interest to a content object. Deliver deterministic behavior as much as possible.

Simple restrictions - Removed overhead created by selectors and replaced them with equality based restrictions.

Protocol Changes - Correctness

Fixed old protocol shortcomings

Loop halting and aggregation - Enable loop halting using hop-limit to fix flaw in nonce-based aggregation.

Type-based names - Enabled meaning in name components without aliasing issues in binary data.

Protocol Changes - Functionality

Increased protocol functionality

Payload in Interests - Efficiently carry data in an interest without requiring the overhead of large names.

Validation Algorithm - Allow the efficient use of variable-length payload for validation algorithms in both Interests and Content Objects.

Protocol Changes - New Features

Introduced new data structures

Manifests - Provide a structure for larger data objects and the transport layer.

Control Messages - A way for the network to communicate about what's happening, return errors, etc.

Protocol Changes - Security

Specified security semantics

Self-certified name restriction - Defined required node behavior to deliver secure self-certified data.

Network caching restriction - Defined required cache behavior to prevent attack amplification.

Protocol Changes - Native

Modified to run over Layer 2

Native Ethernet - Updated protocol to enable CCN to run on the wire.

Fragmentation - Enabled protocol to work under different MTU situations.

Glossary

Interest

A request for a piece of content.

Content Object

A response to an interest. A piece of data, signed.

Message

An Interest or Content Object

Chunk

A Content Object that is part of a larger piece of Data.

Fragment

A network encoded partial CCN Message
Includes static and per-hop-headers

Packet

A network encoded CCN Message (partial or complete)
Includes static and per-hop-headers

Glossary

Name Segment

A labeled name element (“year=2014”)

Name

A ordered set of CCN Name Segments

Name Prefix

A “left” subset of Name Segments in a Name

Content Object Hash

A hash over an encoded Content Object

Similarity Hash

A hash over the matching portion of an Interest

Interest Fragment Identifier

A Similarity Hash used to identify a fragmented Interest

Content Object Fragment Identifier

A Content Object Hash used to identify a fragmented Content Object

Other changes - Summary

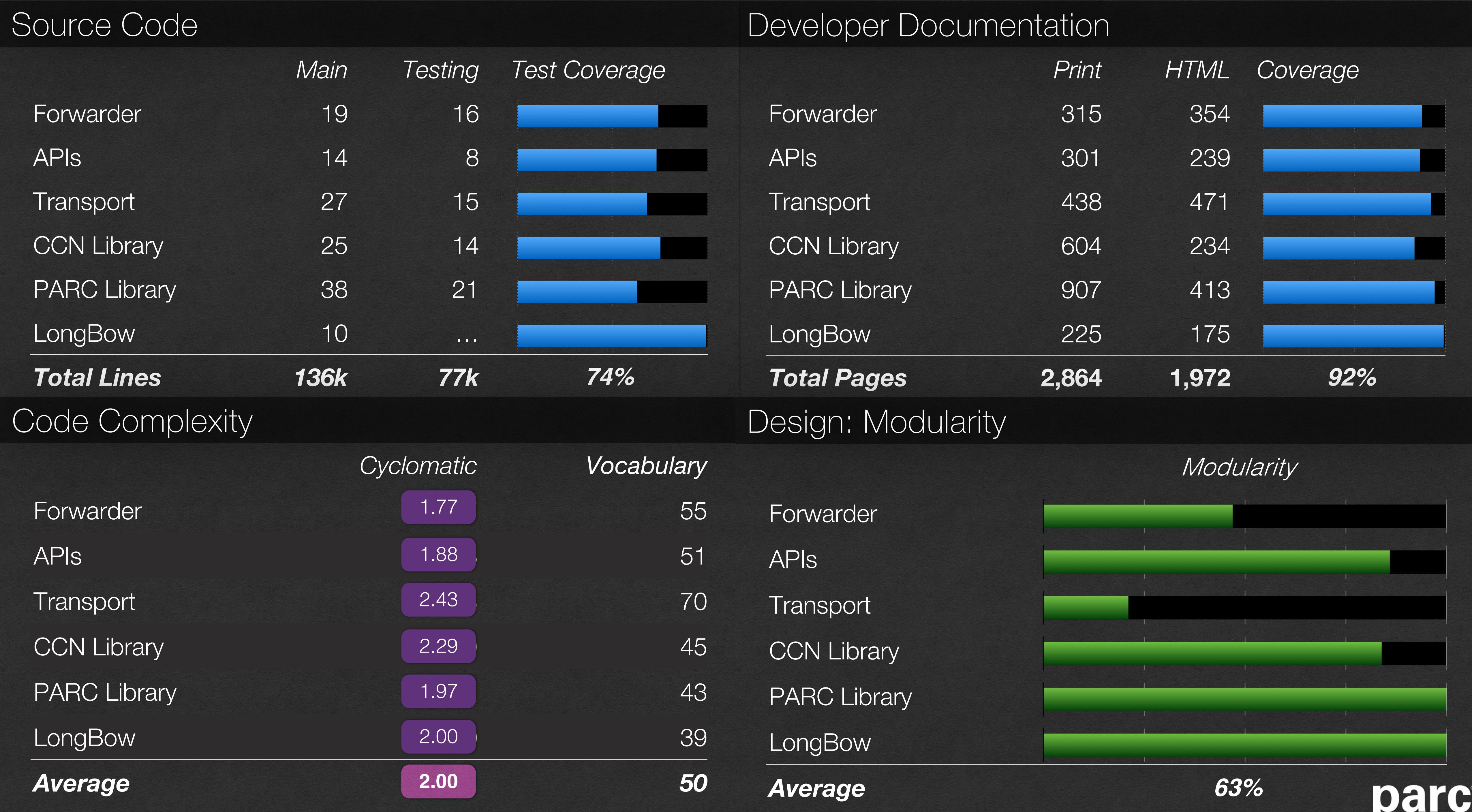
Software (all C)

Coding style

Modular transport

Modular API

Test driven



PRIMARY



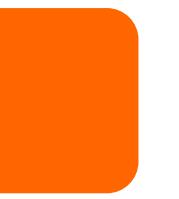
White
R 255
G 255
B 255



PARC blue
R 32
G 84
B 105



Dark blue
R 0
G 35
B 50



Light blue
R 58
G 110
B 143



Orange
R 255
G 102
B 0

Dark gray
R 55
G 55
B 55



Red
R 191
G 29
B 10



Lt. Orange
R 255
G 170
B 10



Lt. Gray
R 213
G 213
B 213



Lt. Green
R 168
G 188
B 50



Faded Blue
R 165
G 255
B 250

Workable area
 $(1920 - 2*83) = 1754\text{px}$

Two column proportional
 $83 + 653 + 1098 + 83$

Body Level One

Body Level Two

Body Level Three

Body Level Four

Body Level Five

Other changes - Summary

Manifest as core object

Control messages

Storage (vs Repo)

Sync as protocol+service

Flow control (vs static pipelining)

CCN - Express

Express Headers

Used for fast forwarding

Similar to CCN enabled MPLS

Can be used together with fragmentation

Simple implementation via hash

Can be done global or local in scope

Forwarding Label

/parc/ccnx/presentations/slides10/v=2/c=0

Globally
Routable
Label

Forwarding Label

/parc/ccnx/presentations/slides10/v=2/c=0

Globally
Routable
Label

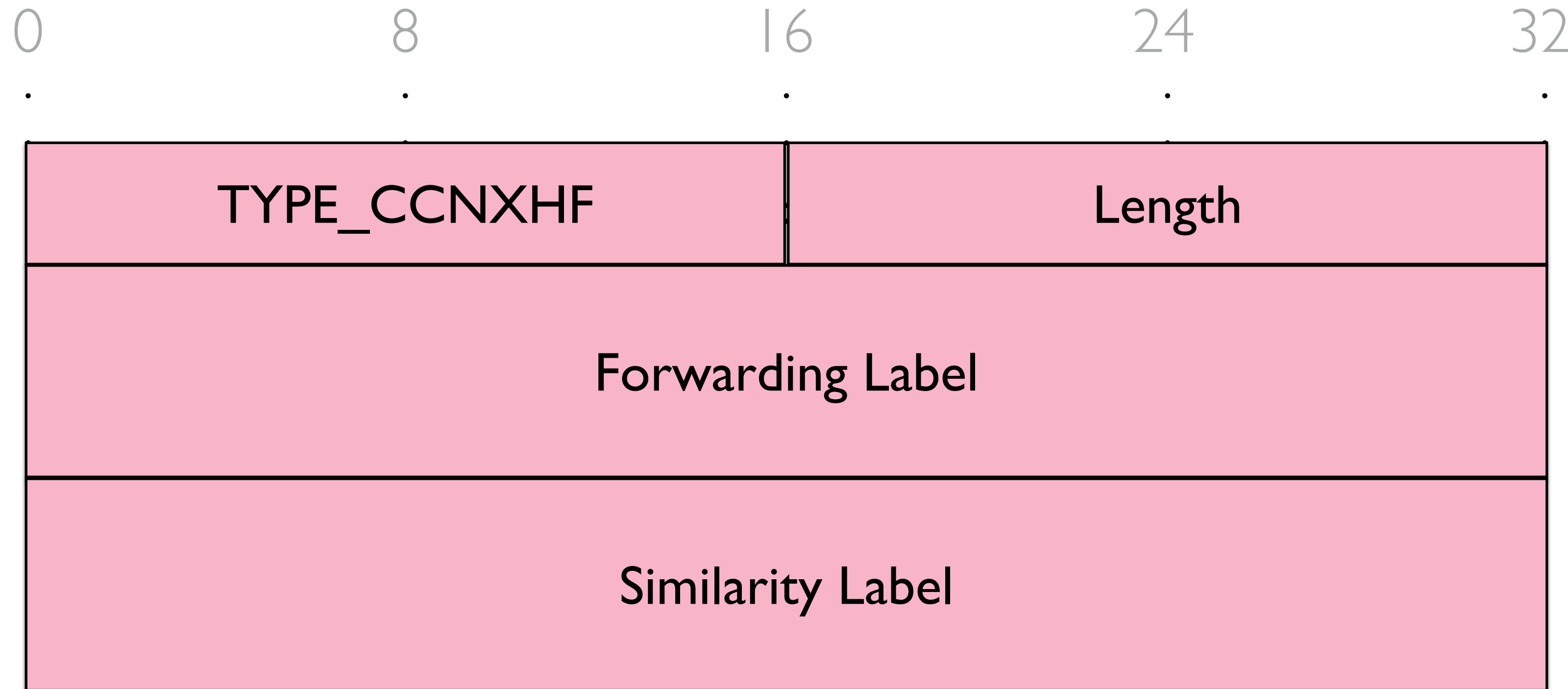
Similarity Label

/parc/ccnx/presentations/slides10/v=2/c=0

- · ?KeyId
- · ?ContentObjectHash

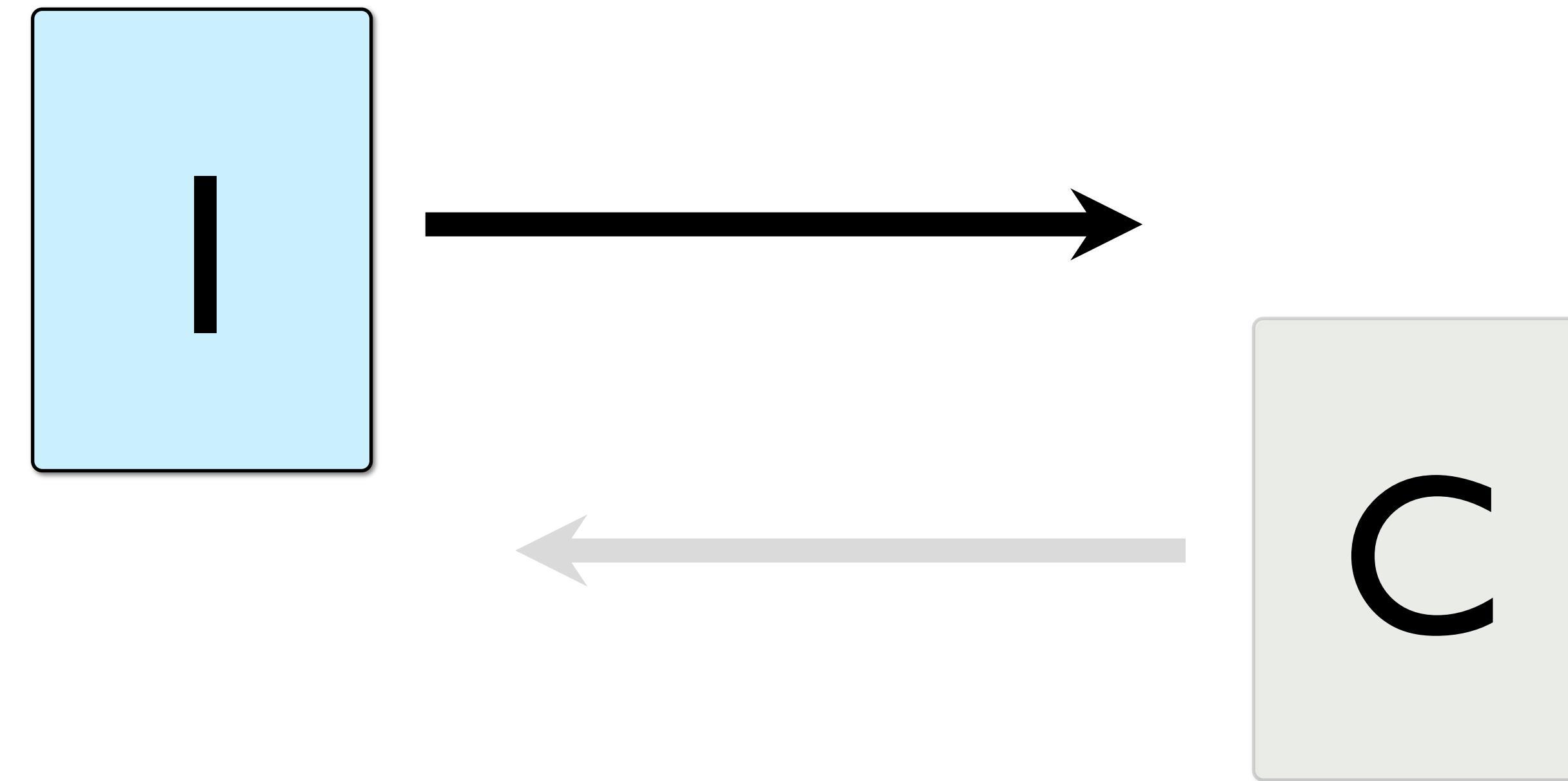
Similarity
Label

Express Headers



CCN - Messages

Interest



Base CCN Message used to request Content Objects

Interest elements

Name

Hierarchical resource identifier. Used for routing and matching.

KeyIdRestriction (optional)

Matching restriction based on a specific key

ContentObjectHashRestriction (optional)

Matching restriction based on a content hash

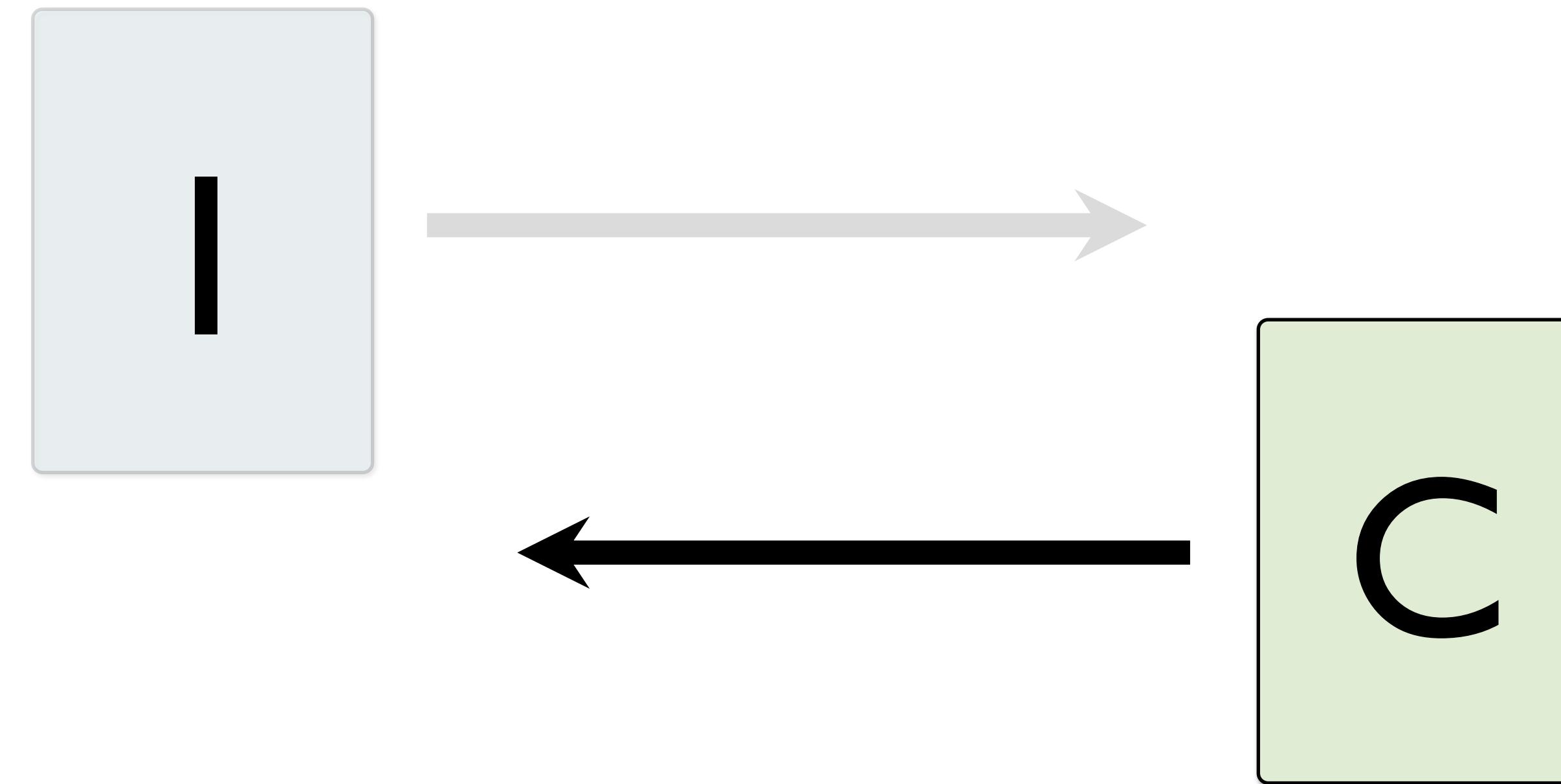
LifetimeRestriction (optional with default value)

Lifetime restriction of the Interest in-network.

Payload (optional)

Data

Content Object



Base CCN Message used to answer an Interest

Content Object elements

Name

Hierarchical resource identifier.

Validation Information (optional)

Validation Algorithm
Validation Information

Payload (optional)

The data.

Metadata (optional)

The data.

Content Object elements

Protocol Information (optional)

Structured information about the content object. Protocols store information here.

Payload

The data.

Signature Block

Cryptographic signature of the Content Object.