NDN-CNL: A Hierarchical Namespace API for Named Data Networking

Jeff Thompson, Jeff Burke, Peter Gusev

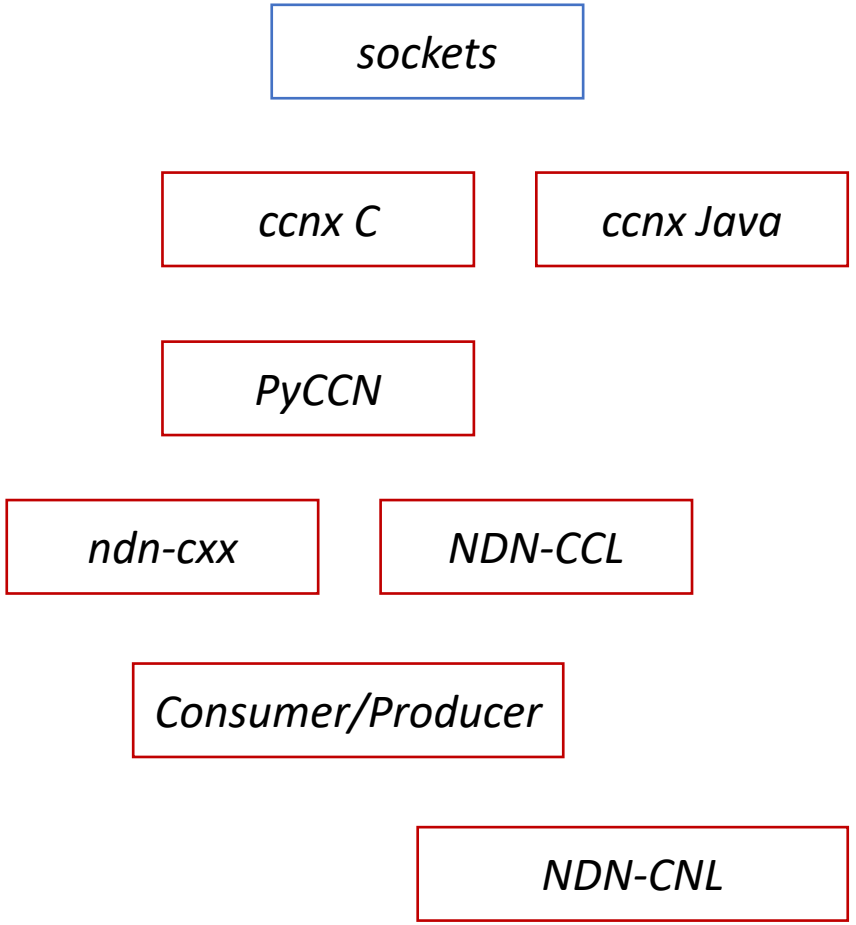
ACM ICN 2019, Macao, China, Sept 24-26

Overview

- NDN Background skipped
- Rationale
- Design
- Implementation
- Examples
- Future Work

Wishes for a new NDN API

- Write data-centric apps **without focusing on Interest/Data** mechanics.
- **Compose data-centric approaches**: segmentation, compound objects, schematized trust, NAC, etc.
- **Incorporate sync** as first-class capability: keep namespaces updated and enable more flexible local manipulations.
- *Align app design with named data design.*



NDN-CNL Goals

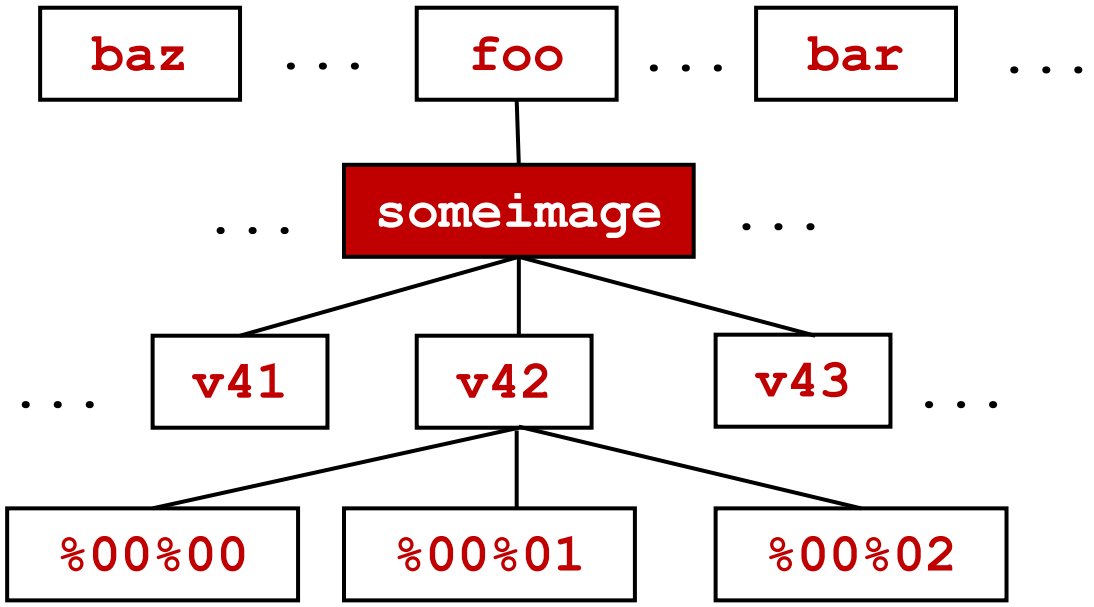
- Provide a collection-oriented interface to NDN data
- Enable consistent manipulation of both app-level objects and data packets
- Employ only a small set of core features
- Minimize loss of generality relative to NDN-CCL

Prefixes often map to ADUs

`/foo/someimage`
mutable image object with a “latest version”

`/foo/someimage/v42`
an immutable version

`/foo/someimage/v42/<segment>`
encoding detail



Producer / Consumer Symmetry

- Both need to be told the name of new objects
- Vary in ways to learn a name
  - Create a new object
  - Have part of a name and construct the rest
  - Overhear a name in a packet
  - Receive announcements over name sync

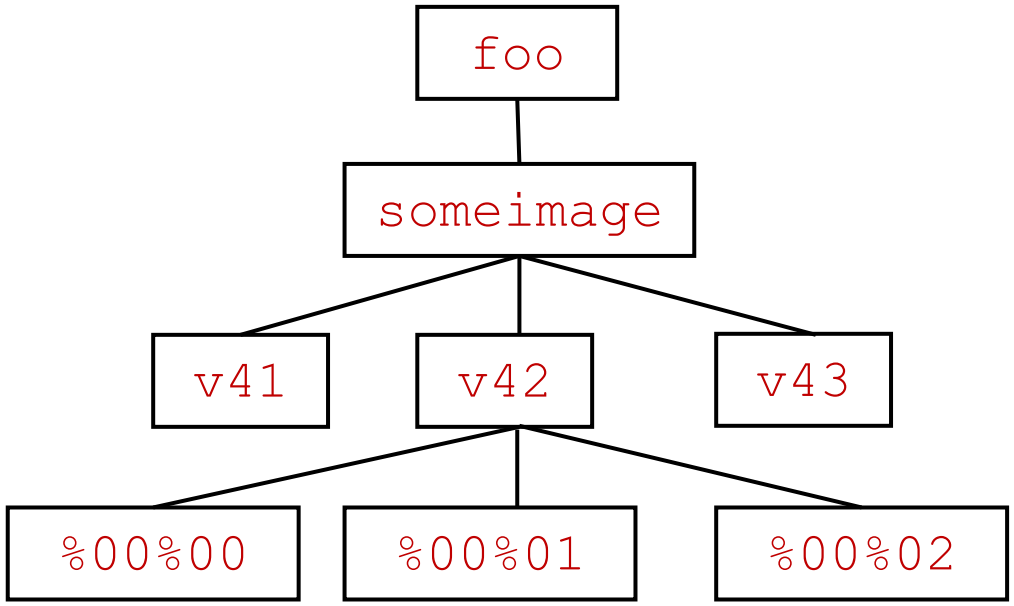
Always wanted to enumerate names…

- Then, could apply wildcard/regexp matching
- And borrow from query languages like XPath (W3C std)

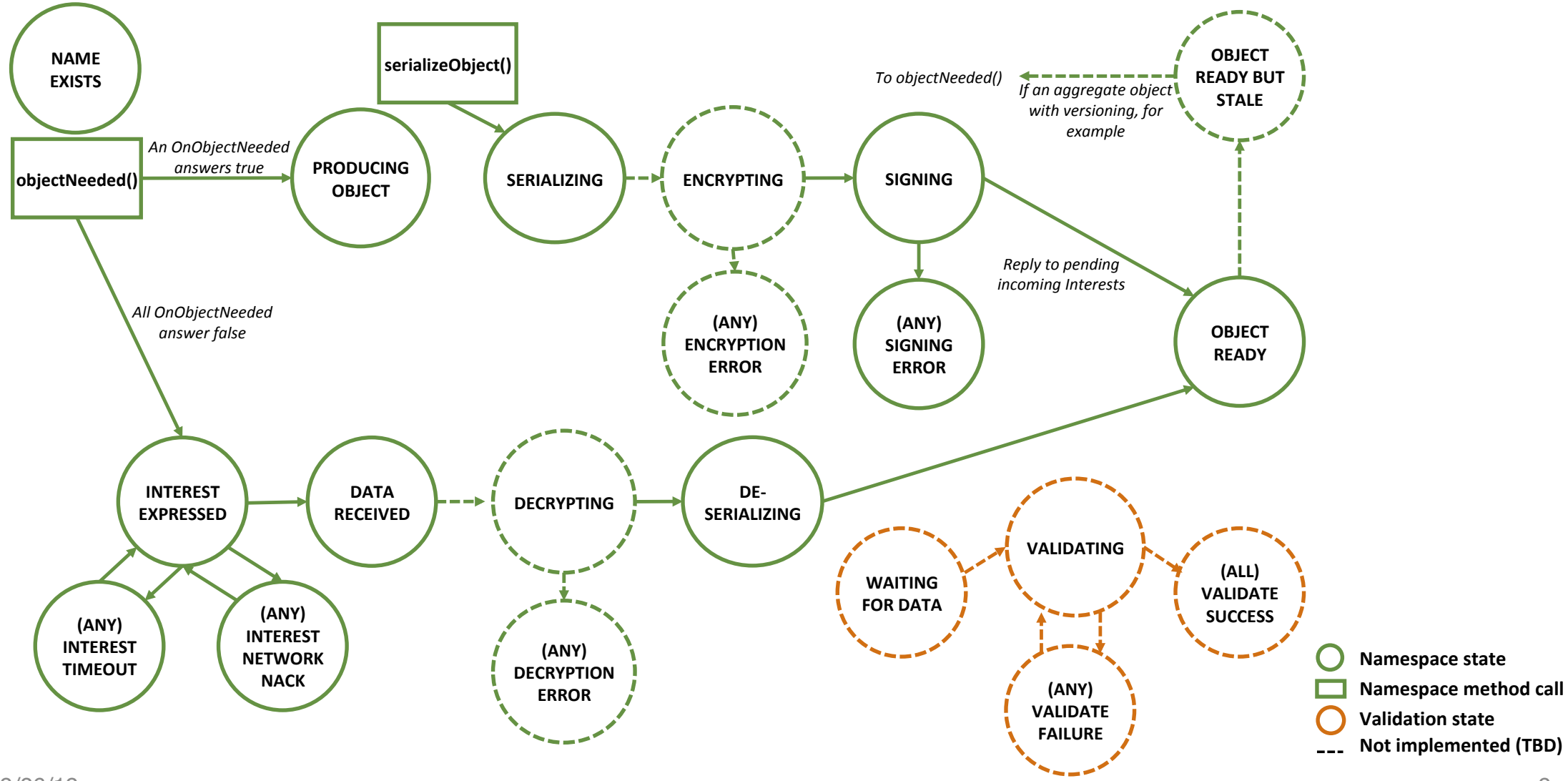
Path Expression	Result
/bookstore/book[1]	Selects the first book element that is the child of the bookstore element. **Note:** In IE 5,6,7,8,9 first node is[0], but according to W3C, it is [1]. To solve this problem in IE, set the SelectionLanguage to XPath: *In JavaScript: xml*.setProperty("SelectionLanguage","XPath");
/bookstore/book[last()]	Selects the last book element that is the child of the bookstore element
/bookstore/book[last()-1]	Selects the last but one book element that is the child of the bookstore element
/bookstore/book[position()<3]	Selects the first two book elements that are children of the bookstore element

CNL Approach: *Namespace* API

- Apps manipulate a local tree of names. Changes propagated to/from the net.
- Apps interact with nodes using application data structures deserialized from network objects (strings, dictionaries, etc.) or packet-granularity payloads.
- Each node can have handlers that serialize/deserialize, sign/verify, encrypt/decrypt its children.
- Asynchronous programming model, with common states managed by the library for consistency and simplicity.



Use a common state machine to enhance composability

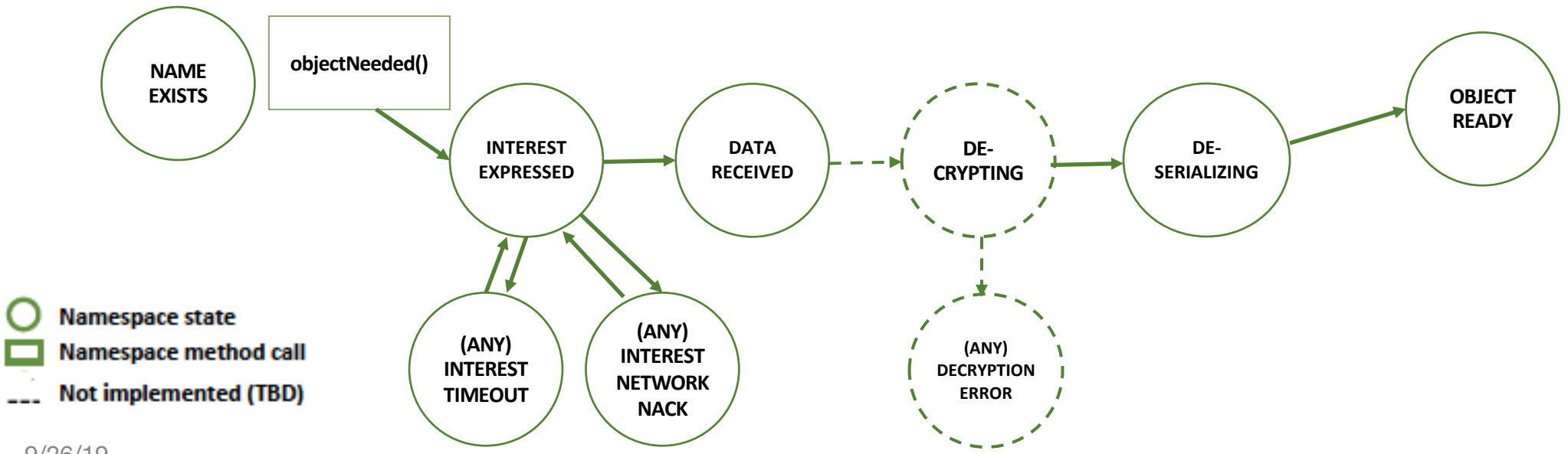


Simple Example – Fetch Segmented Content

```
face = Face()

image = Namespace("/foo/someimage/42") # immutable version 42
image.setFace(face)

def onSegmentedObject(handler, obj):
    print("Got image")

SegmentedObjectHandler(image, onSegmentedObject).objectNeeded()
```

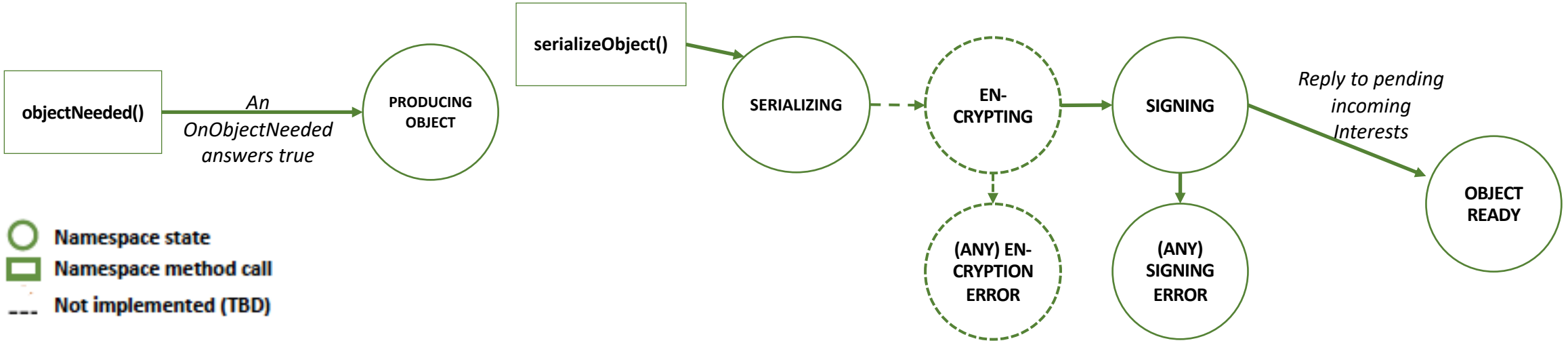
Typical Consumer

1. App registers to respond to state change, calls objectNeeded()
2. CNL sends Interest and receives Data
3. CNL attaches Data to the node, sets state to OBJECT_READY
4. App gets state changed callback for OBJECT_READY



Producer

1. App registers to respond to objectNeeded() on a node (or subtree)
2. CNL receives an Interest, calls objectNeeded() on its name node
3. App responds that it can produce the object, CNL waits
4. App attaches Data packets the Namespace => OBJECT_READY
5. CNL uses attached Data packets to reply to pending interests



Unified Consumer and Producer

- CCL/cxx: consume with expressInterest(), produce with registerPrefix()
- Unified: objectNeeded()
  - If the app calls objectNeeded(), it is a consumer
  - If the app responds to objectNeeded(), it is a producer
  - If the Namespace tree already has the object, it acts as a cache
- Apps can employ the Namespace as:
  - *Cache* - CNL receives an Interest; Namespace already has immutable Data attached; CNL replies
  - *Workspace* - One part of app calls objectNeeded(), another part produces and attaches Data

Handlers

- Assigned to a prefix node to handle child Data packets
- For app, provide structured application objects
  - E.g., generalized object ContentMetaInfo
  - “Object ready”, not “Data packets ready”
- For network, execute naming and payload conventions
  - Segmented content, versioned objects, latest data retrieval, serialize/deserialize
- Goal: Composability, supporting mulitiple handlers to the same node
  - E.g., one for segmenting, another for application-specific serialization
  - More easily support security: “Mix in” a standardsecurity handler with other handlers

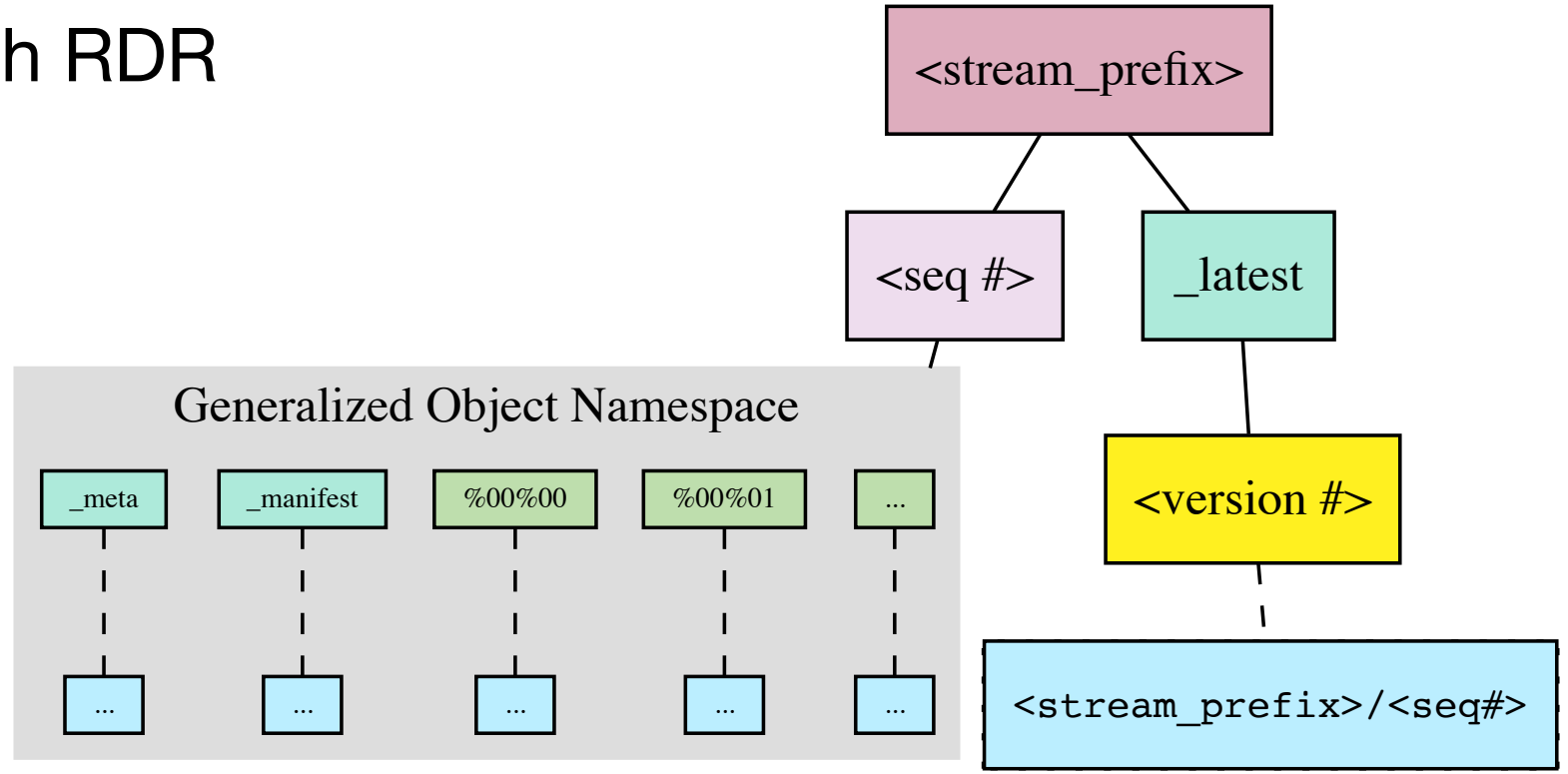
Implementation Examples

Implementation

- Built on the Common Client Library (CCL)
- Implemented in C++ and Python
  - Heavy use of callbacks (standard mechanism for each language)
- Applied in AR, video streaming, repo sync apps
- Experience lead us to a single callback for all node state changes

Generalized Object Stream

- Real-time Data Retrieval (RDR) with _latest packet
- Fixed-size Interest pipeline in current impl.
- If timeout, restart with RDR



GObjStream Producer

```
face = Face()
keyChain = KeyChain()
face.setCommandSigningInfo(keyChain,
  keyChain.getDefaultCertificateName())

stream = Namespace("/ndn/stream/run/28/annotations", keyChain)
stream.setFace(face,
  lambda prefix: print("Register failed: " + prefix.toUri()))

handler = GeneralizedObjectStreamHandler(stream)
handler.addObject(Blob(”Payload 1"), "text/html")
handler.addObject(Blob(”Payload 2"), "text/html”)
…
```

GObjStream Consumer

```
face = Face()
stream = Namespace("/ndn/stream/run/28/annotations")
stream.setFace(face)

def onNewObject(seqNumber, contentMetaInfo, objectNamespace):
    print("Got seq# " + str(seqNumber) + ": " +
          str(objectNamespace.obj))

GeneralizedObjectStreamHandler(stream, 10, onNewObject).objectNeeded()
```

Many-to-Many Namespace Updates w/ Sync

- CNL Namespace API supports sync + local search
- Currently implemented: PSync
- Enable sync on a node in the Namespace tree to a certain depth
  - Depth limitation: E.g., announce new versions, but not child segments
- Repo usage integrated
  - Producer joins repo sync namespace, announces names to be fetched / stored

PSync Example

```
face = Face()
keyChain = KeyChain()
face.setCommandSigningInfo(keyChain, keyChain.getDefaultCertificateName())

applicationPrefix = Namespace("/test/app", keyChain)
applicationPrefix.setFace(face)
userPrefix = applicationPrefix["alice"] # or "bob"

def onStateChanged(nameSpace, changedNamespace, state, callbackId):
    if (state == NamespaceState.NAME_EXISTS and
          not userPrefix.name.isPrefixOf(changedNamespace.name)):
        print("Received " + changedNamespace.name.toUri())
applicationPrefix.addOnStateChanged(onStateChanged)
applicationPrefix.enableSync()
userPrefix["v1"]._setObject(Blob("content1"))
userPrefix["v2"]._setObject(Blob("content2"))
```

Local Eval of Wildcards on Sync'd Names

```
applicationPrefix = Namespace(Name("/test/app/users"), keyChain)

applicationPrefix.setFace(face,

    lambda prefix: dump("Register failed for prefix", prefix))

applicationPrefix.enableSync()  # Sync with other instances using this
namespace

# ... Since the Namespace object childComponents is iterable, enumerate simply
elsewhere -

regex = re.compile("Bob.*")

for child in filter(lambda c: regex.match(str(c)),
applicationPrefix.childComponents):

    applicationPrefix[child].objectNeeded(True)  # generate interests to
retrieve
```

Name-based Access Control

- The API for NAC defines DecryptorV2
  - Key chain with consumer's private key
- Consumer calls Namespace method setDecryptor(decryptor)
- If supplied, the decryptor is used in the state machine
- See example to add encryption to the SegmentedObjectHandler:
  - https://github.com/named-data/PyCNL/blob/master/examples/test_nac_producer.py
  - https://github.com/named-data/PyCNL/blob/master/examples/test_nac_consumer.py

Future Work

- How to best propagate packet-level events to higher-level objects?
  - timeouts, validation failure, expired freshness, etc.
- How to combine handlers and prevent/identify conflict
- Storage integration
  - Optimize performance as a memory content cache
  - Integrated persistent storage/repo functionality
  - “Swap to disk” of content to save memory
- Maintain statistics on higher-level prefix nodes
  - Interest retransmission, RTT, segment fetching progress/rate

Thank you!

- Thanks to Lixia Zhang and Alex Afanasyev for input on the CNL
- Thanks to Ashlesh Gawande for help integrating Psync
- Thanks to our shepherd, John Wroclawski

- Code
  - Python: https://github.com/named-data/PyCNL
  - C++: https://github.com/named-data/cnl-cpp