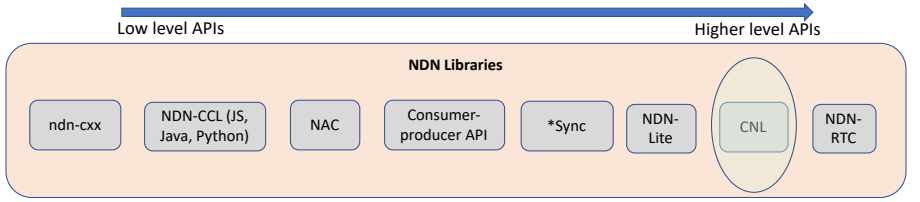
Namespace-focused APIs for NDN

NDN-CNL, NTSchema

NDN-CNL: Jeff Thompson, **Jeff Burke**, Peter Gusev

NTSchema: Xinyu Ma

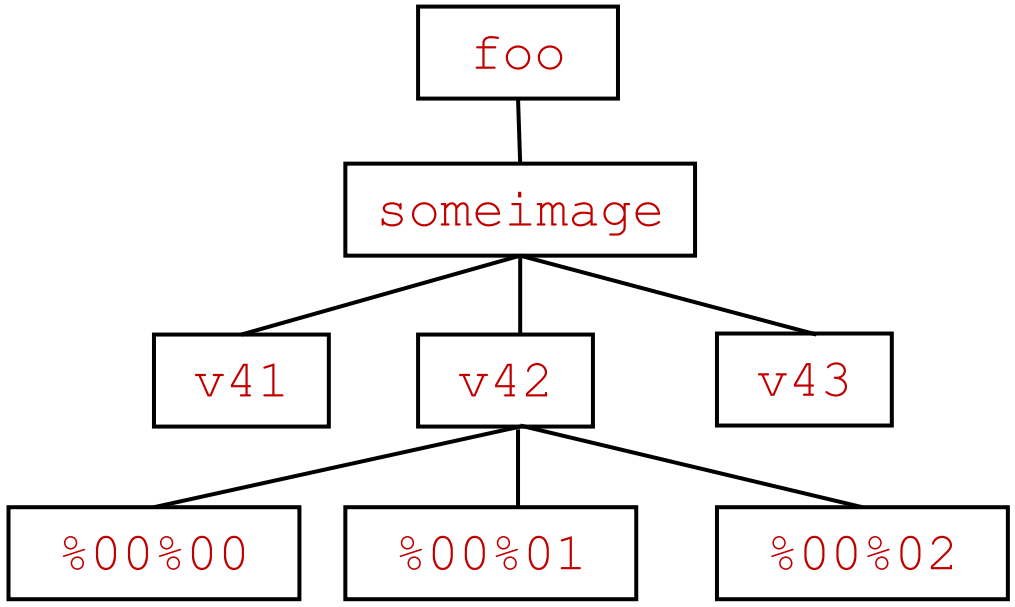
General APIs for Named Data **Apps**



- CCNX C: Wire format (~2009)
- CCNX Java: Content object & utility abstractions (~2011)
- CCL / CXX: Interest/data exchange
  - Support for schematized trust, name-based access control, etc.
  - Descendants such as NDNts: Modern language features
- Consumer/Producer: High-level fetching/publishing patterns
  - Socket-like Interface (no relation among sockets)
- Pub-Sub libraries (~2019)
  - Natural extension, treating prefixes as topics
- CNL: Namespace API (~2019)
  - Organize async networking via the namespace itself (C++, Python; arbitrary namespaces)
  - NTSchema - app framework (Python; static namespaces)

Objectives

- **Align app design with named data design.**
- Write data-centric apps **without focusing on Interest/Data** mechanics.
- **Compose data-centric approaches**: segmentation, versioning, compound objects, schematized trust and access control, pub-sub behavior, etc.
- **Incorporate sync** as first-class capability: keep high-level namespaces updated and enable flexible local operations.



NDN Common Name Library

"NDN-CNL: A Hierarchical Namespace API for [NDN]", *ACM ICN 2019*.

- First realization of a concept; not intended to be the last.
- In-memory namespace representation maintained by the library for the application.
- API provides consistent manipulation of both app-level objects and data packets, and symmetry between producers and consumers.
- Compose and apply handlers to namespace subtrees.
- Employ only a small set of core features.
- Minimize loss of generality relative to NDN-CCL
  (NT Schema will focus further.)

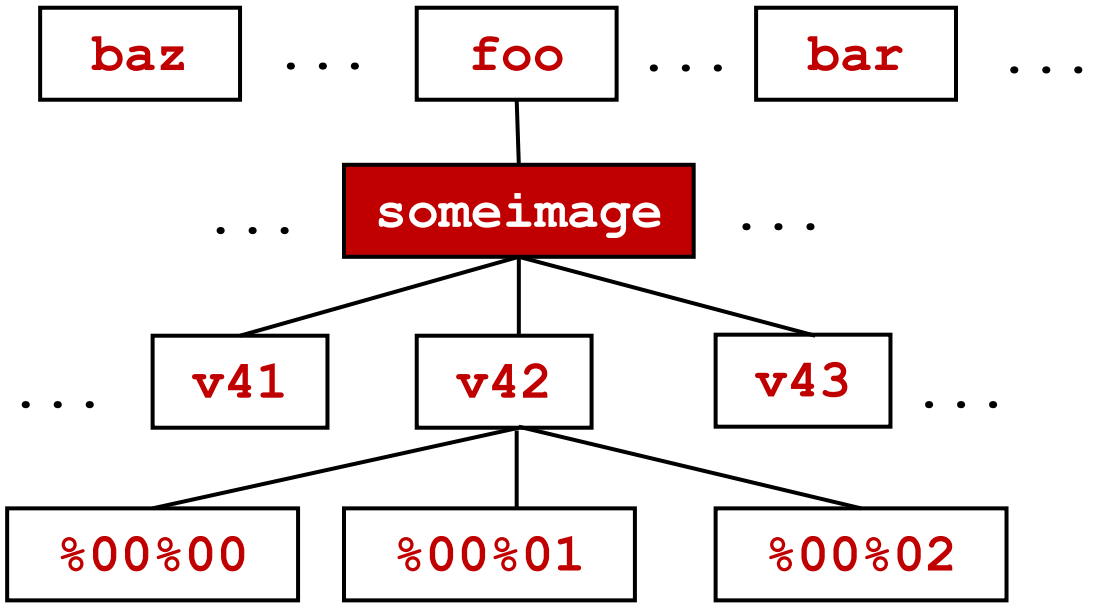
github.com/named-data/cnl-cpp
github.com/named-data/PyCNL

Map higher-level abstractions onto prefixes

/foo/someimage
mutable image object (stream) with a “latest version”

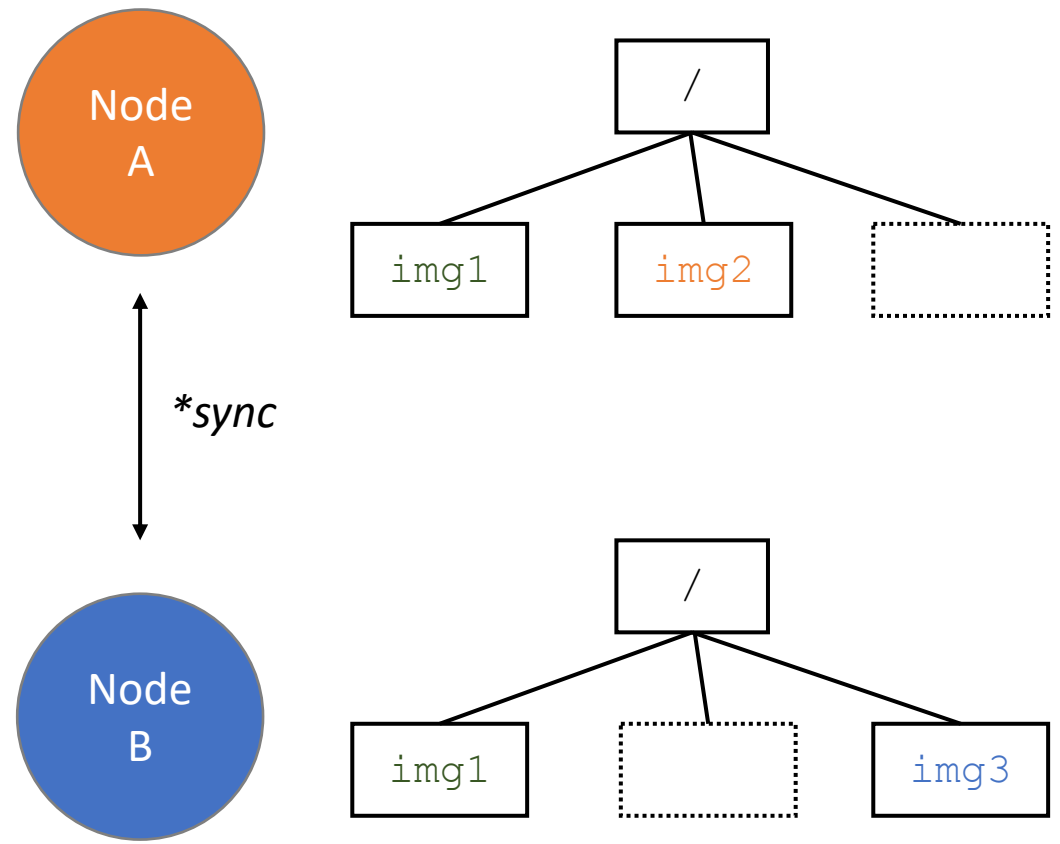
/foo/someimage/v42
an immutable version

/foo/someimage/v42/
packet



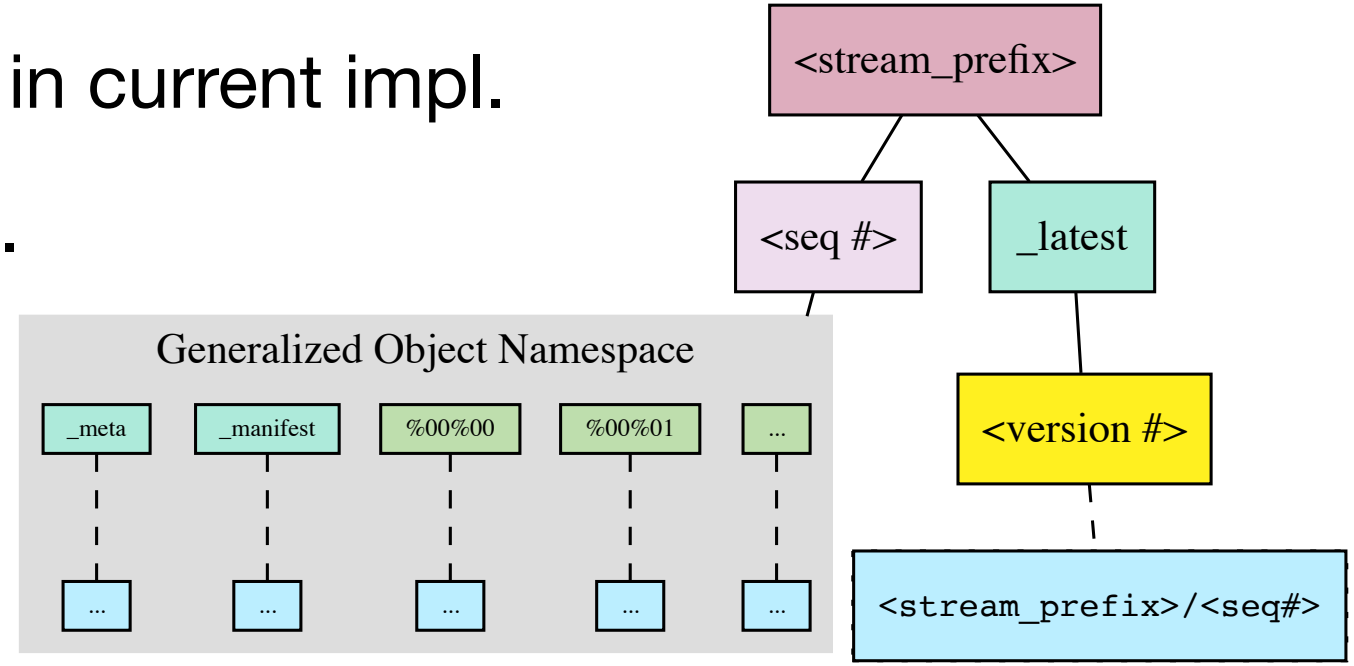
Multiparty Comm. w/Sync

- TCP = two-party communications (usually).
- For NDN, *multiparty communications about a namespace* should be the norm.
- CNL apps can sync relevant namespaces to their preferred depth.
- App code notified of new names, and then handlers for the names decide what to do.
  (How to schematize handlers is challenging: NTSchema explores how.)
- Keeping namespace in-memory allows local namespace functions not available directly from the network: enumeration, search, etc.



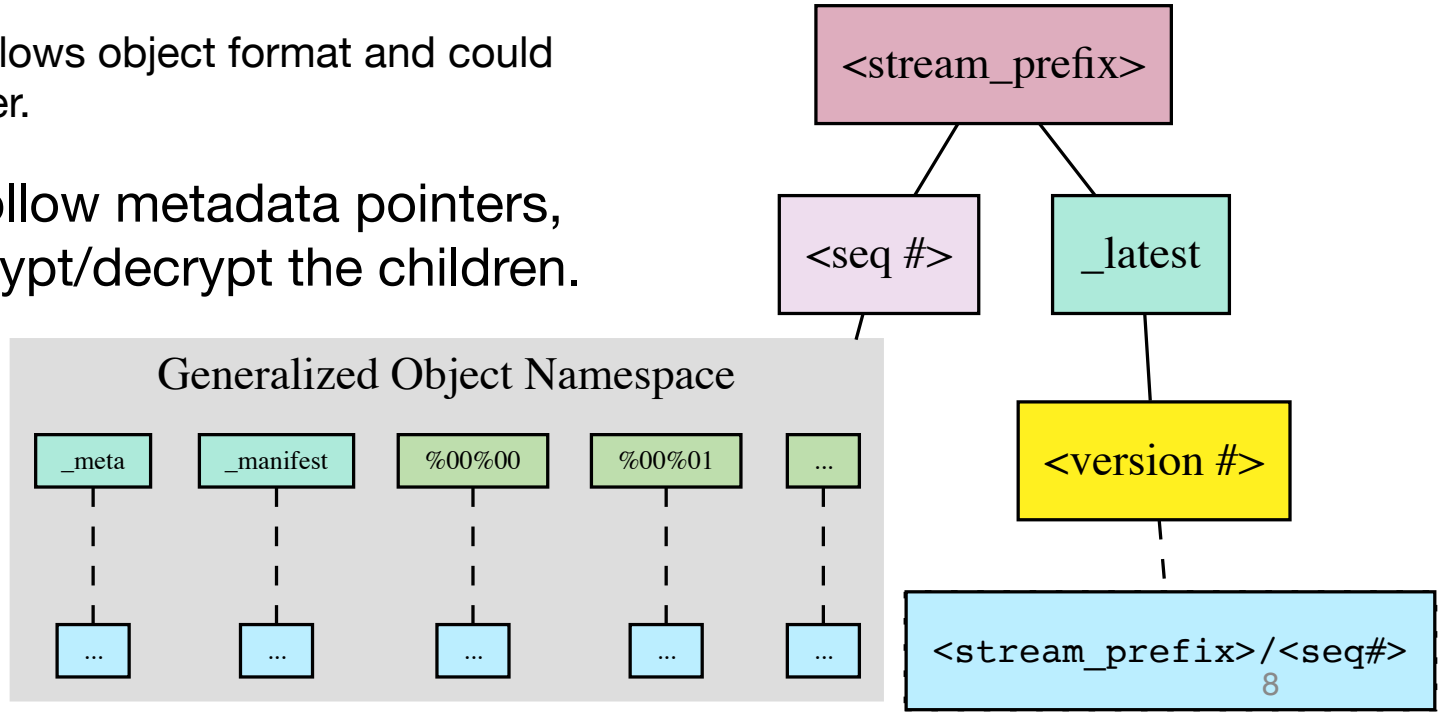
Generalized Object Stream

- Stream of general object schema created for our group's apps.
- Good use case: versions, sequences, metadata, at two layers.
- Real-time Data Retrieval (RDR) with _latest packet
- Fixed-size Interest pipeline in current impl.
- If timeout, restart with RDR.



CNL's approach to the Generalized Obj Stream

- Apps manipulate <stream_prefix> or any intermediate child in a local tree of names, descending all the way to packet detail for objects whose data is known.
- Data serialized, cached; new names propagated via sync.
- For prefix-level data, apps interact with application data structures. For example, subclassed from general stream and object handlers.
  - For example, NDN-RTC video stream follows object format and could be manipulated by a CNL general handler.
- Each node can have handlers that follow metadata pointers, serialize/deserialize, sign/verify, encrypt/decrypt the children.
- CNL has an asynchronous model, with common states managed by the library for consistency and simplicity.



GObjStream Producer

```
// Above: initialize and select keychain.
stream = Namespace("/ndn/stream/run/28/annotations", keyChain)
…
handler = GeneralizedObjectStreamHandler(stream)
handler.addObject(Blob(”Payload 1"), "text/html")
handler.addObject(Blob(”Payload 2"), "text/html”)
…
```

Consumer

```
def onNewObject(seqNumber, contentMetaInfo, objectNamespace):
    print("Got seq# " + str(seqNumber) + ": " +
          str(objectNamespace.obj))
GeneralizedObjectStreamHandler(stream, 10, onNewObject).objectNeeded()
```

TouchNDN

```
from pyndn import Name, MetaInfo
from pyndn.util import Blob
from pyndn.util.common import Common
from pycnl import Namespace
from pycnl.generalized_object import GeneralizedObjectHandler
from _process_ndn import face, keyChain
import preview_trigger
from preview_trigger import readPreview

handler = None

def setupPreviewHandler():
    global previewVersion, handler

    previewObject = Namespace(me.parent().path, keyChain)
    def onRegisterFailed(prefix):
        print('registration failure for prefix', prefix.toUri())
    def onregisterSucceed(prefix, clbkId):
        print('registration success for prefix', prefix.toUri())

    previewObject.setFace(face, onRegisterFailed, onregisterSucceed)
    handler = GeneralizedObjectHandler()
    handler.setMaxSegmentPayloadLength(8000)
    metaInfo = MetaInfo()
    metaInfo.setFreshnessPeriod(op('timer1').par.length*1000)

    def onObjectNeeded(namespace, neededNamespace, callbackId):
        print('got request for the preview ', neededNamespace.getName().toUri())
        if not (neededNamespace is previewObject):
            return False

        print('version',preview_trigger.previewVersion)
        print(Name.Component.fromVersion(preview_trigger.previewVersion).toEscapedString())
        versionNamespace = previewObject[Name.Component.fromVersion(preview_trigger.previewVers
        versionNamespace.setNewDataMetaInfo(metaInfo)
        previewData = Blob(readPreview())
        print('will publish', previewData.size(), 'bytes')
        handler.setObject(versionNamespace, previewData, "image/jpeg")
```

_ndn_handlers


```
def onObjectNeeded(namespace, neededNamespace, callbackId):
    print('got request for the preview ', neededNamespace.getName().toUri())
    if not (neededNamespace is previewObject):
        return False

    print('version',preview_trigger.previewVersion)
    print(Name.Component.fromVersion(preview_trigger.previewVersion).toEscaped
    versionNamespace = previewObject[Name.Component.fromVersion(preview_trigg
    versionNamespace.setNewDataMetaInfo(metaInfo)
    previewData = Blob(readPreview())
    print('will publish', previewData.size(), 'bytes')
    handler.setObject(versionNamespace, previewData, "image/jpeg")
```

rs

github.com/remap/TouchNDN

Challenges / Future work

- Dynamic namespaces make things complicated
  - Need some protobuf / versec style of formalizing / sharing namespace schema.

- Composability is easier to describe than implement
  - Developer-level configuration of details not schematized

- NTSchema attempts to address this, with some simplifications
  - Static / well-known namespace
  - No sync yet

Thompson et al. “NDN-CNL: A Hierarchical Namespace API for [NDN]”, *ACM ICN 2019*.

github.com/named-data/cnl-cpp
github.com/named-data/PyCNL

Thank you!

jburke@remap.ucla.edu