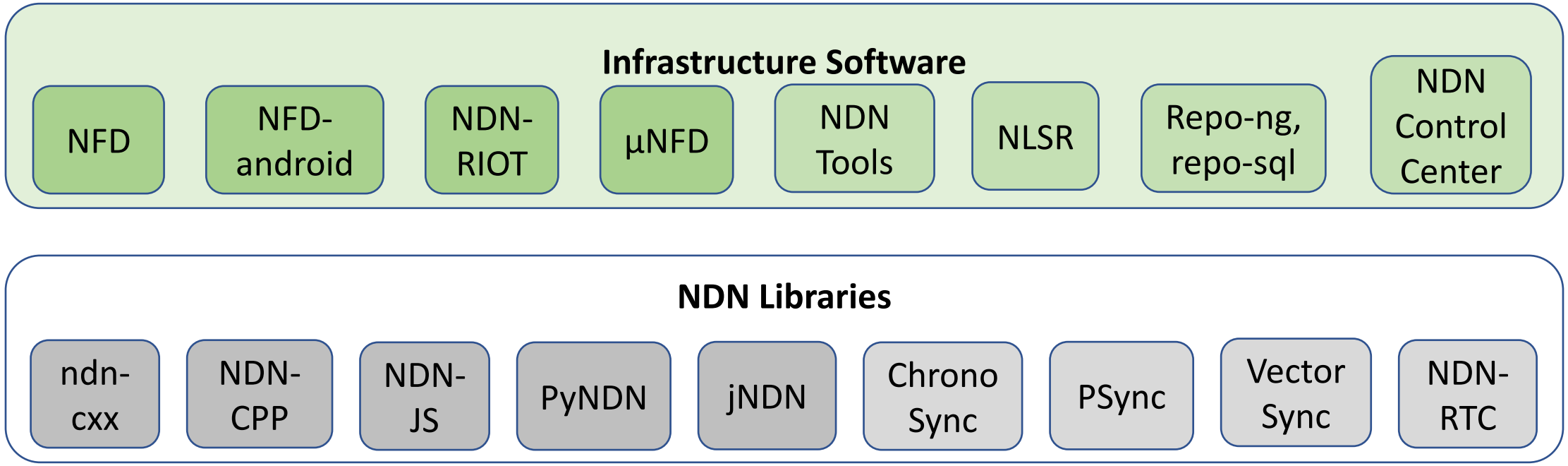
ACM ICN 2018 Tutorial
Boston, MA, September 21, 2018

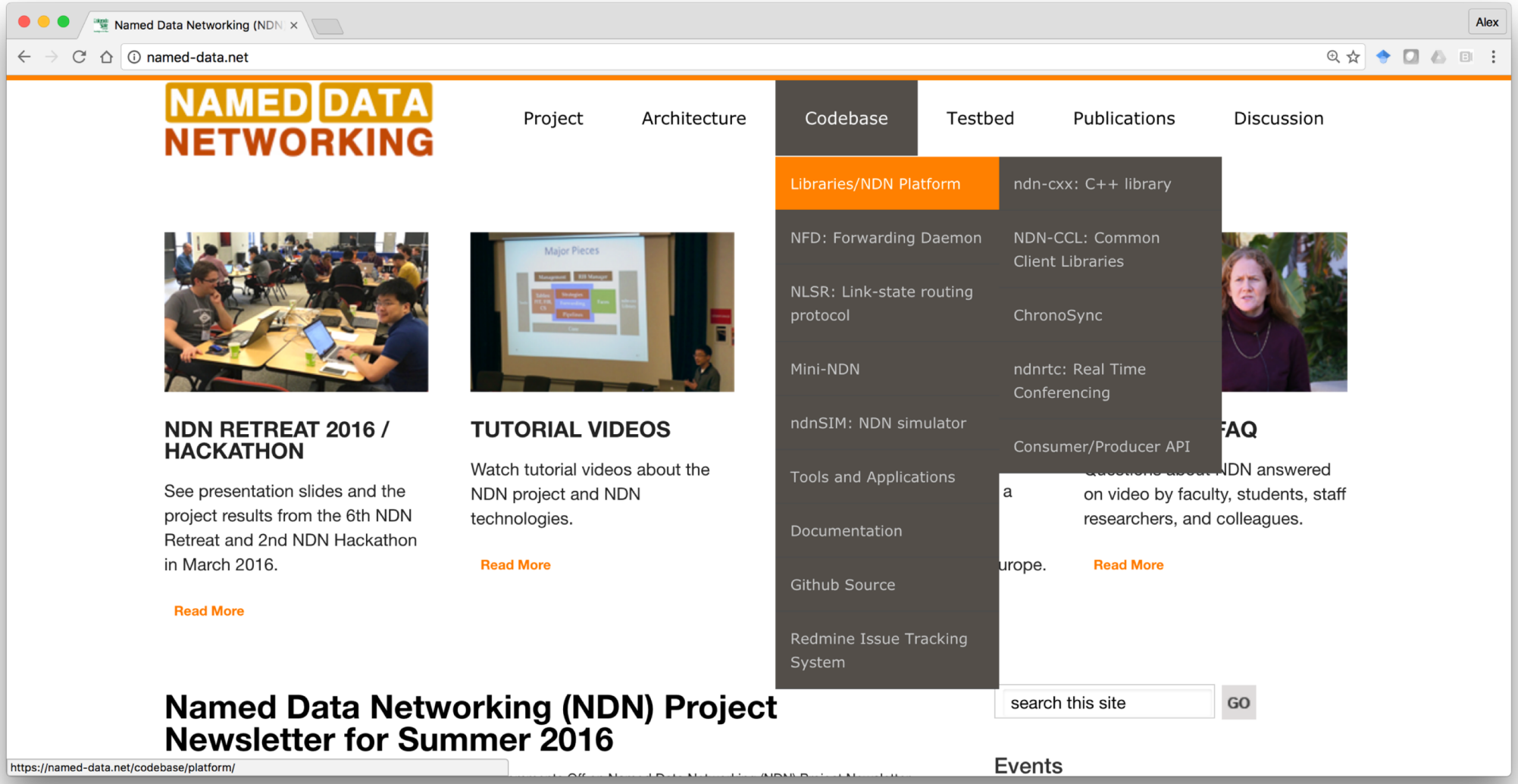
New Library Directions

Jeff Thompson, Alex Afanaysev

NDN Codebase Overview

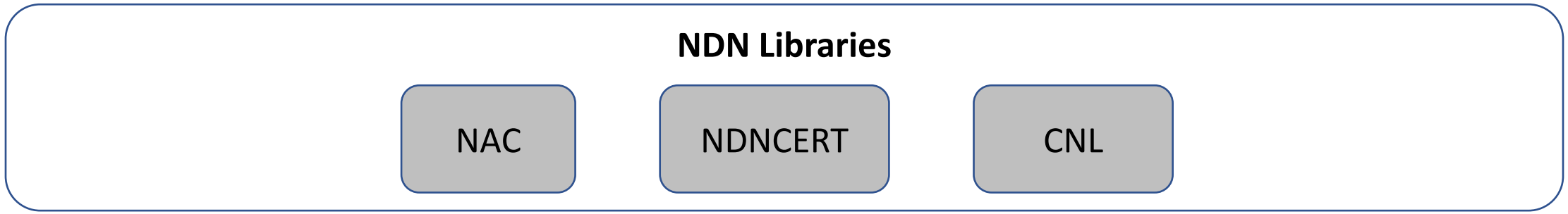


Starting Point: https://named-data.net/ ➠ Codebase



New Libraries

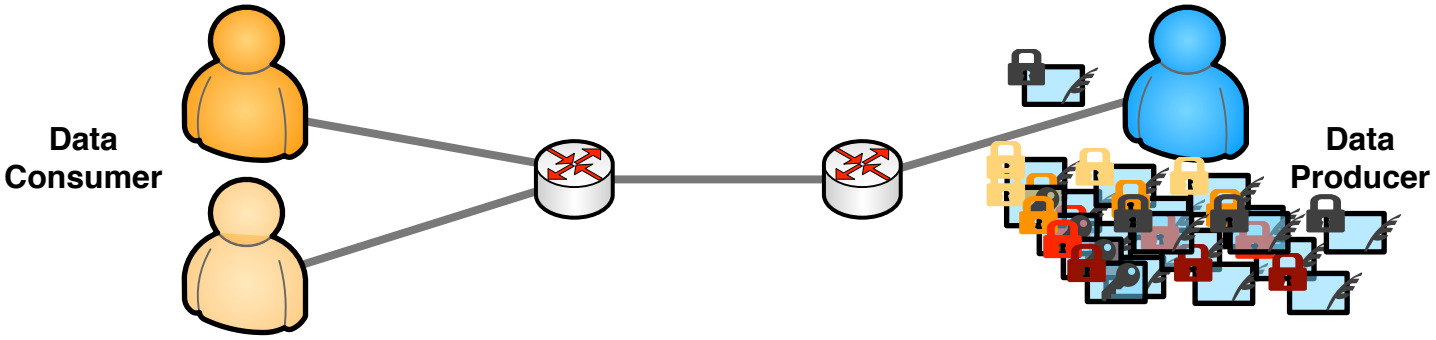
- Security
  - Name-Based Access Control
  - NDNCERT
  - (Expanded on in the afternoon)
- Applications
  - Common Name Library: motivation and directions



Data-centric confidentiality

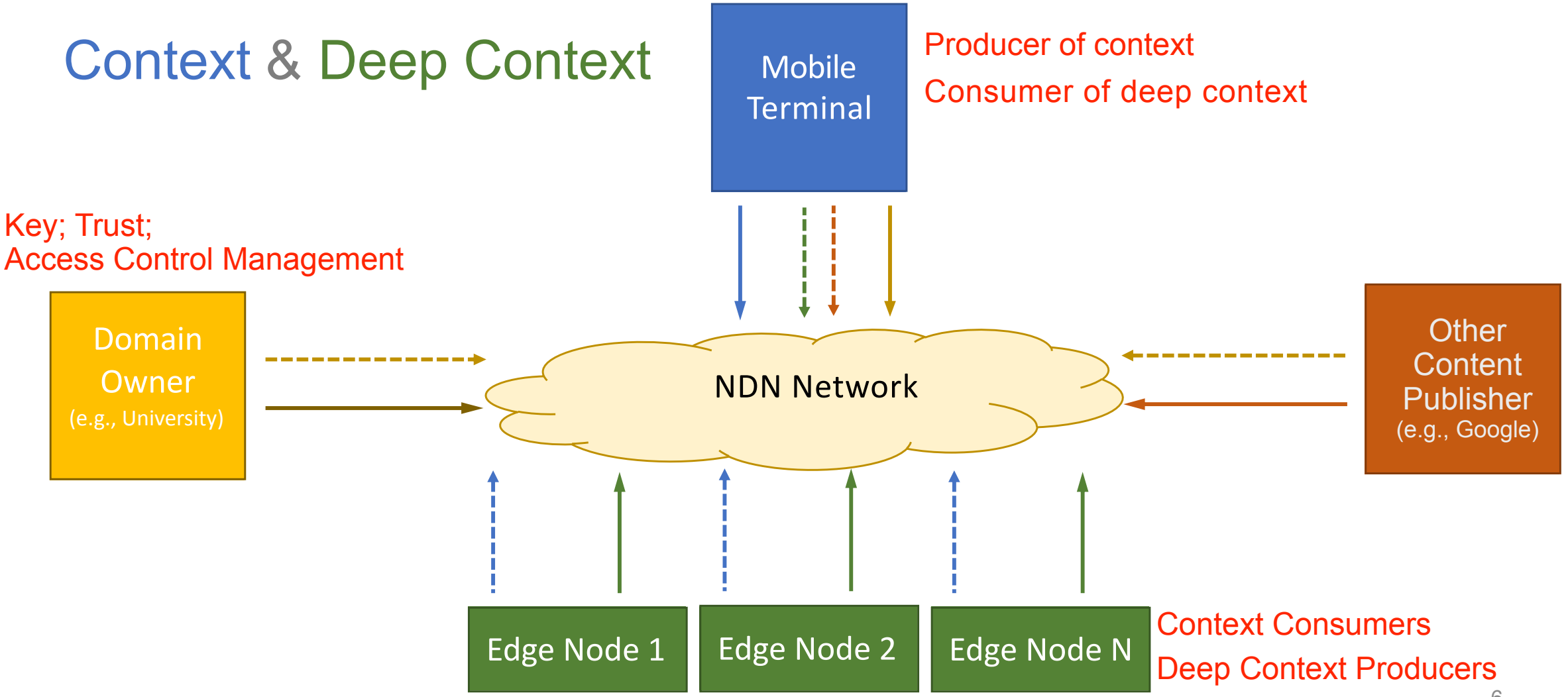
Think how to secure data, not channels where data is transferred

- Encrypt data at the time of production
- Distribute decryption keys to authorized consumers
- Design challenges
  - How does a producer learn the authorized consumers?
    - changing authorized consumers
    - distributed production
  - How to distribute decryption keys efficiently?



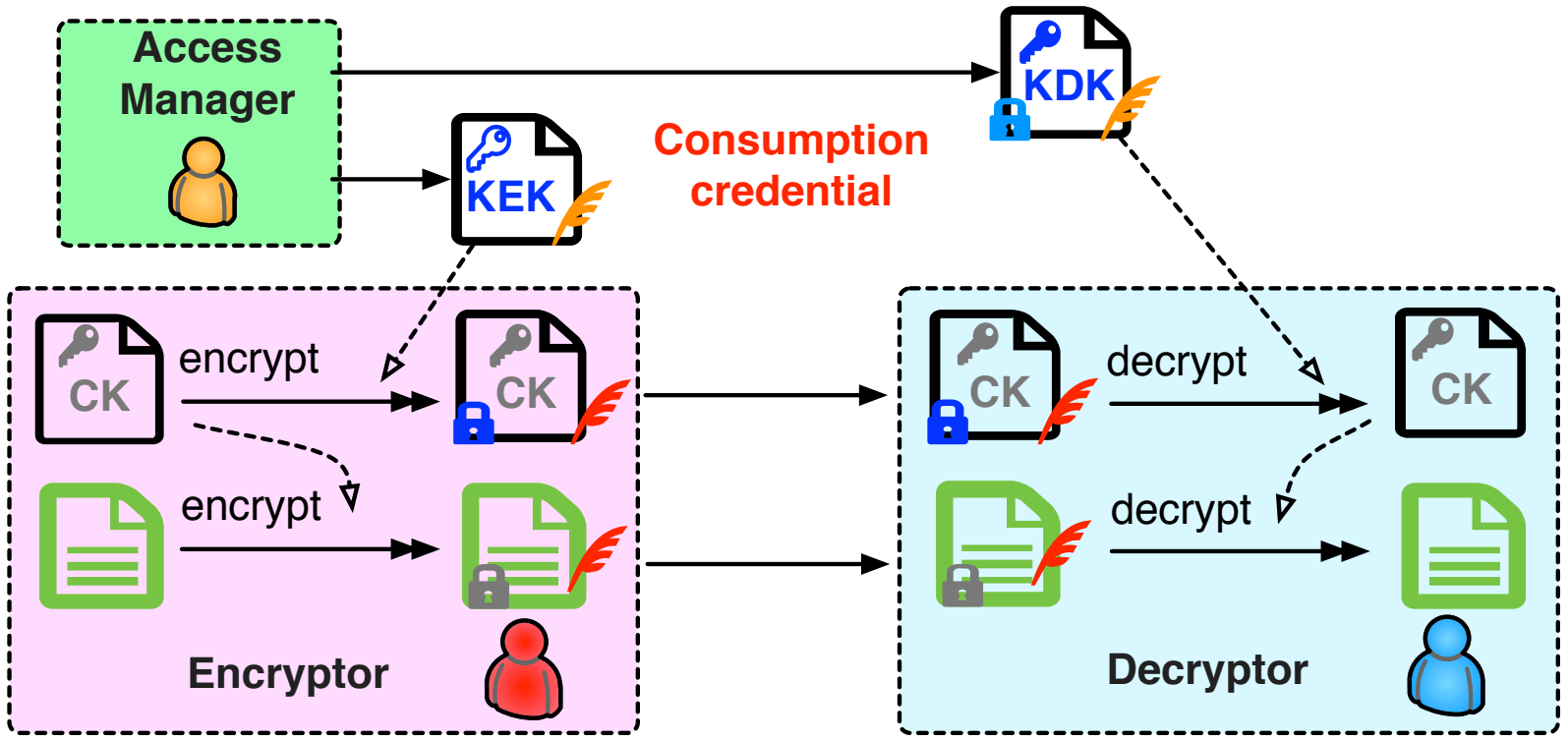
Augmented Reality Environment

Context & Deep Context



Name-Based Access Control (Concepts)

(Data Owner) Entity that control access to the data associated with the namespace



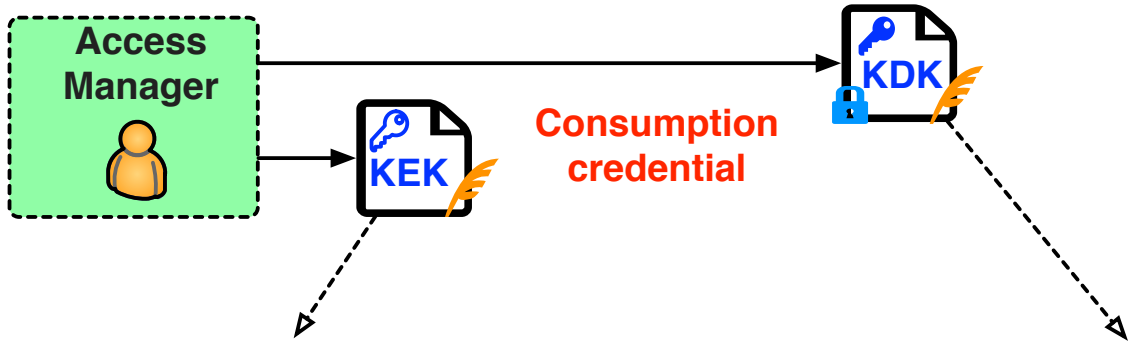
(Producer) Entity that encrypts data based on namespace association

(Consumer) Entity that decrypts data based on namespace association

Managing Access

Authorized to create policies = owns identity
</mypolicy/subset>/KEY/<key-id>

Defines policy via RSA key
</mypolicy/subset>/NAC/<granularity>/KEY/<key-id>



Realizes encryption policy using a public key

</mydata/subset>/NAC /<granularity>/KEK/<key-id>

Realizes deceryption policies with encrypted version of private key (KDK)

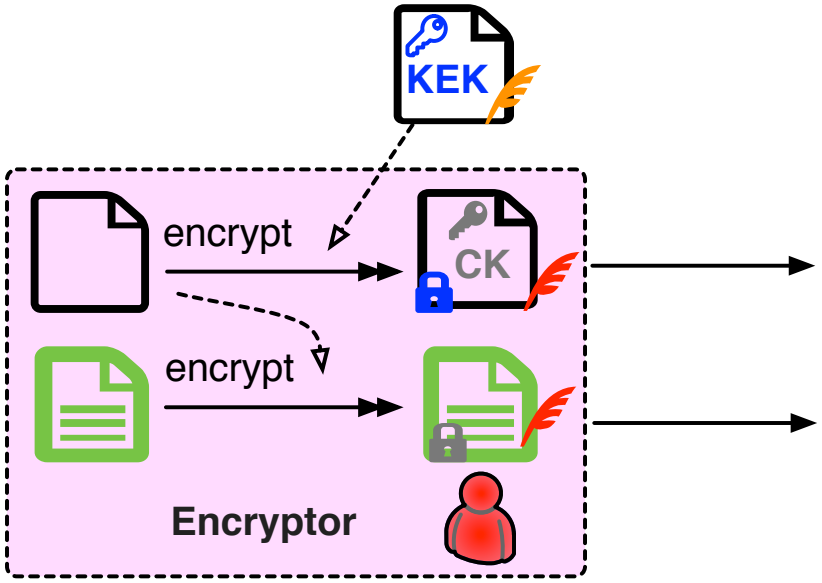
</mydata/subset>/NAC /<granularity>/KDK/<key-id> /ENCRYPTED-BY/<user-X-prefix>/KEY/<key-id>
…
</mydata/subset>/NAC /<granularity>/KDK/<key-id> /ENCRYPTED-BY/<user-Y-prefix>/KEY/<key-id>

Producing and Encrypting Data

From Access Manager / provisioned or dedicated data owner storage

Executes named policy via fetching corresponding KEK

</policy/subset>/NAC/<granularity>/KEK/<key-id>



Generates (re-generates) symmetric Content Key (CK)
Publishes CK data under configured namespace, encrypted by KEK

</ckdata/prefix>/CK/<key-id>/ENCRYPTED-BY/</mydata/subset>/NAC/KEK/<key-id>

Encrypts input data using CK, returns encrypted content
Exact name of the corresponding CK data is embedded in the encrypted content

Receiving and Decrypting Data

From Access Manager / provisioned or dedicated data owner storage

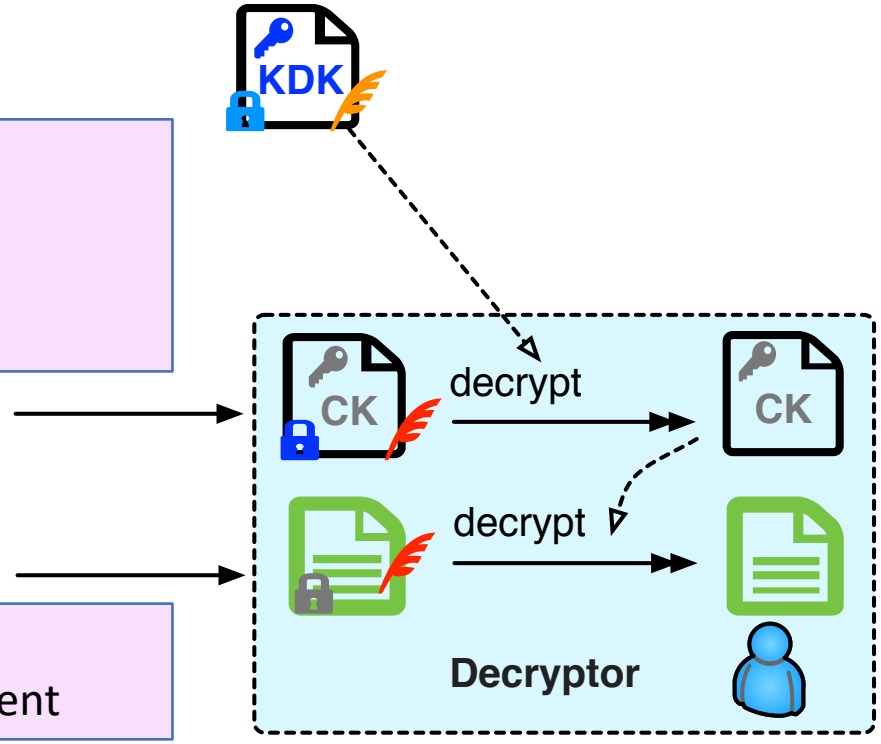
Executes decryption policy via fetching/failing to fetch KDK

</mydata/subset>/NAC/KDK/<key-id>/ENCRYPTED-BY/<user-identity>/KEY/<key-id>

Fetches CK data for the name extracted from input encrypted payload

</ckdata/prefix>/CK/<key-id>
…/ENCRYPTED-BY</mydata/subset>/NAC/KEK/<key-id>

Decrypts the input data using CK, returns encrypted content
Exact name of the corresponding CK data is embedded in the encrypted content



NAC Library API Highlighss

```
#include "access-manager.hpp"

...

AccessManager accessManager(identity, granularity, ...);

accessManager.addMember(authorizedCert1);
accessManager.addMember(authorizedCert2);
```

```
Encryptor encrypto(accessPolicyName, ckName, ...);

Data data(dataName);
data.setFreshnessPeriod(10 _s);

auto content = encryptor.encrypt(data, dataSize);
data.etContent(content.wireEncode());

keyChain.sign(data);
```

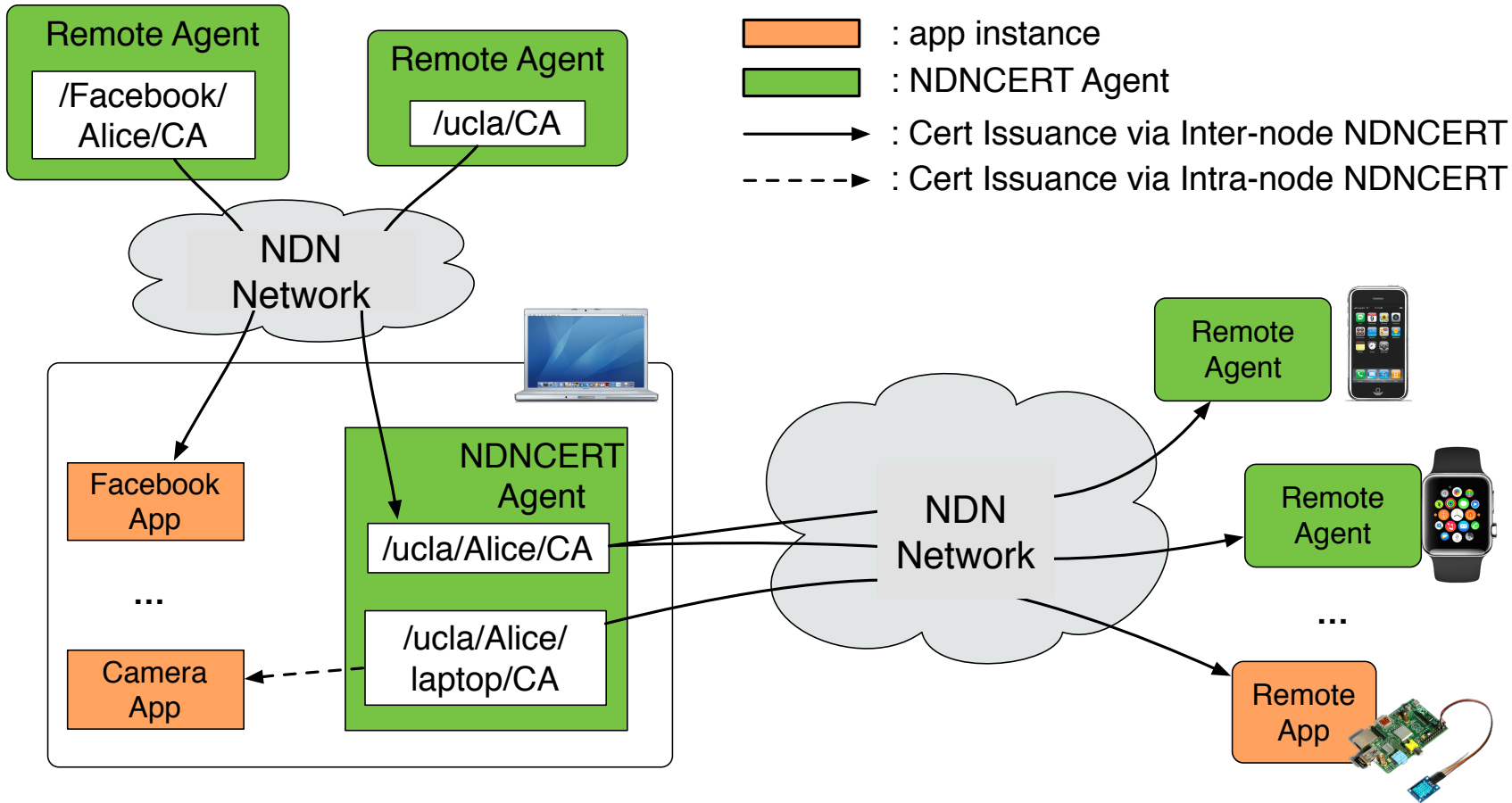
```
Decryptor decryptor(identity, ...);

decryptor.decrypt(data.getContent().blockFromValue(),
  [=] (ConstBufferPtr content) {
    ...
  },
  [=] (const ErrorCode&, const std::string& error) {
    std::cerr << "Cannot decrypt data: " << error << std::endl;
});
```

NAC Next Steps

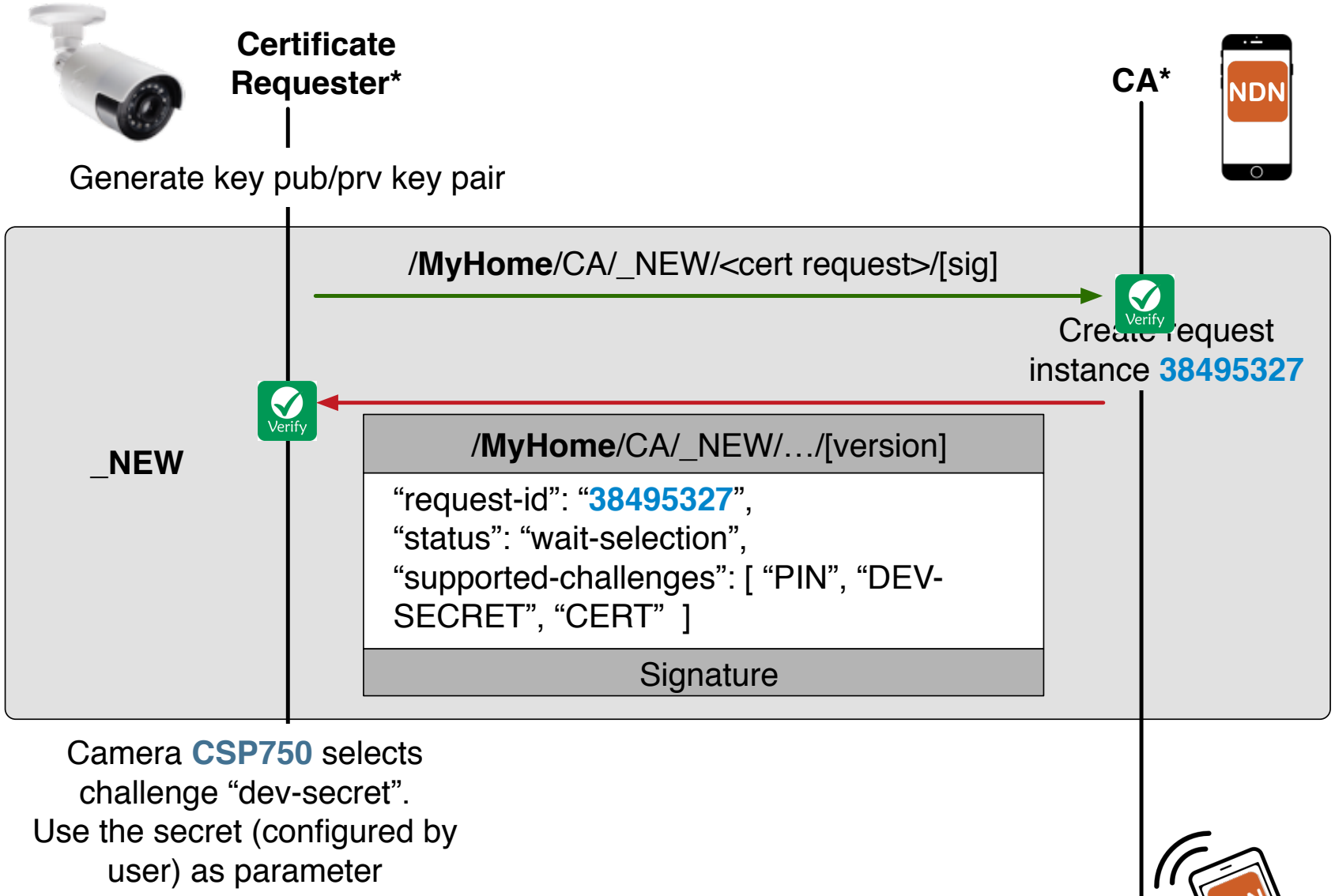
- Integrate attribute-based encryption
  - Currently available as a a separate experimental library
- Integrate in more application prototypes to refine APIs and features

NDNCERT: Certificate Management

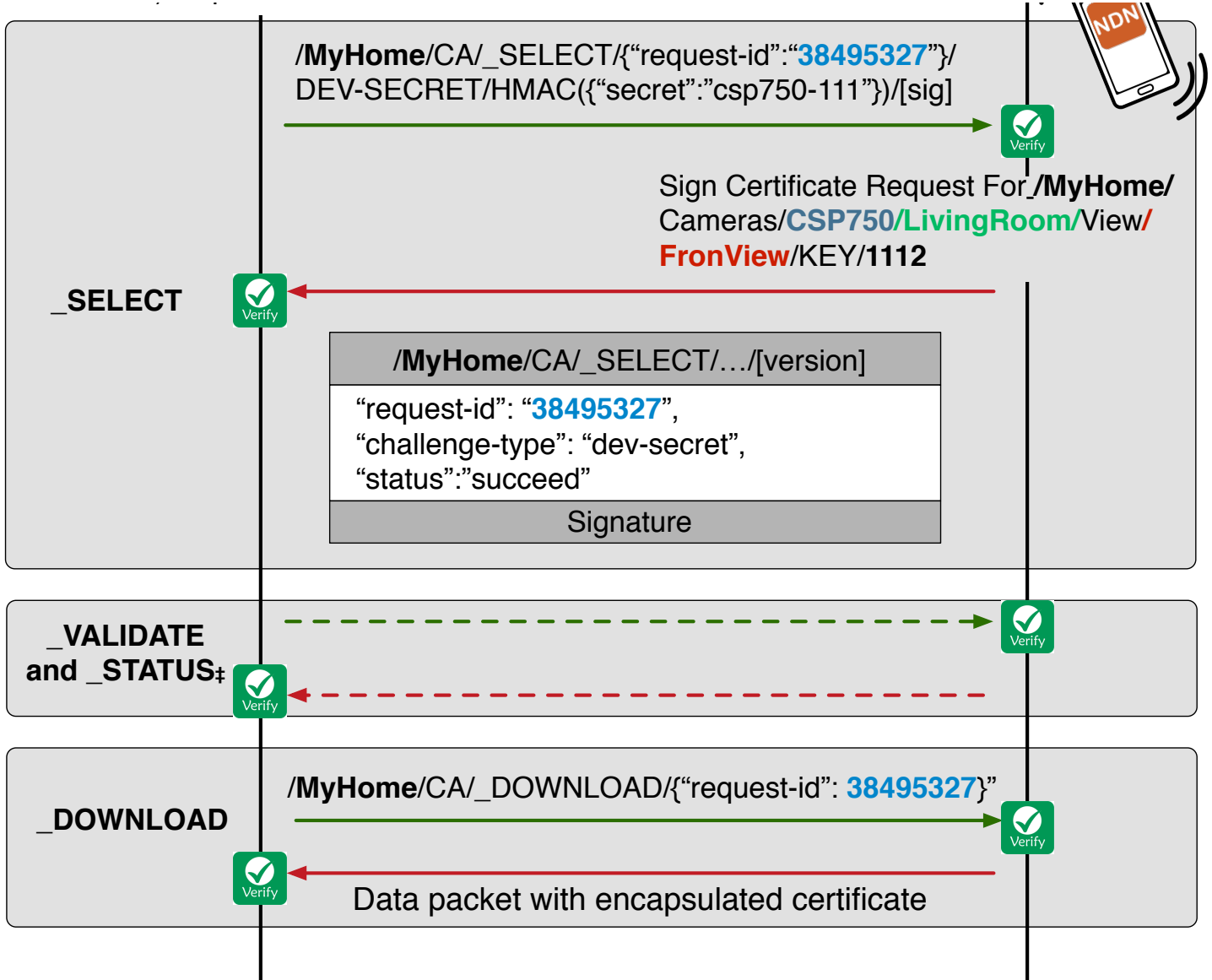
- Streamlined certificate request and issuance
- Any node can act as a local CA for the namespace



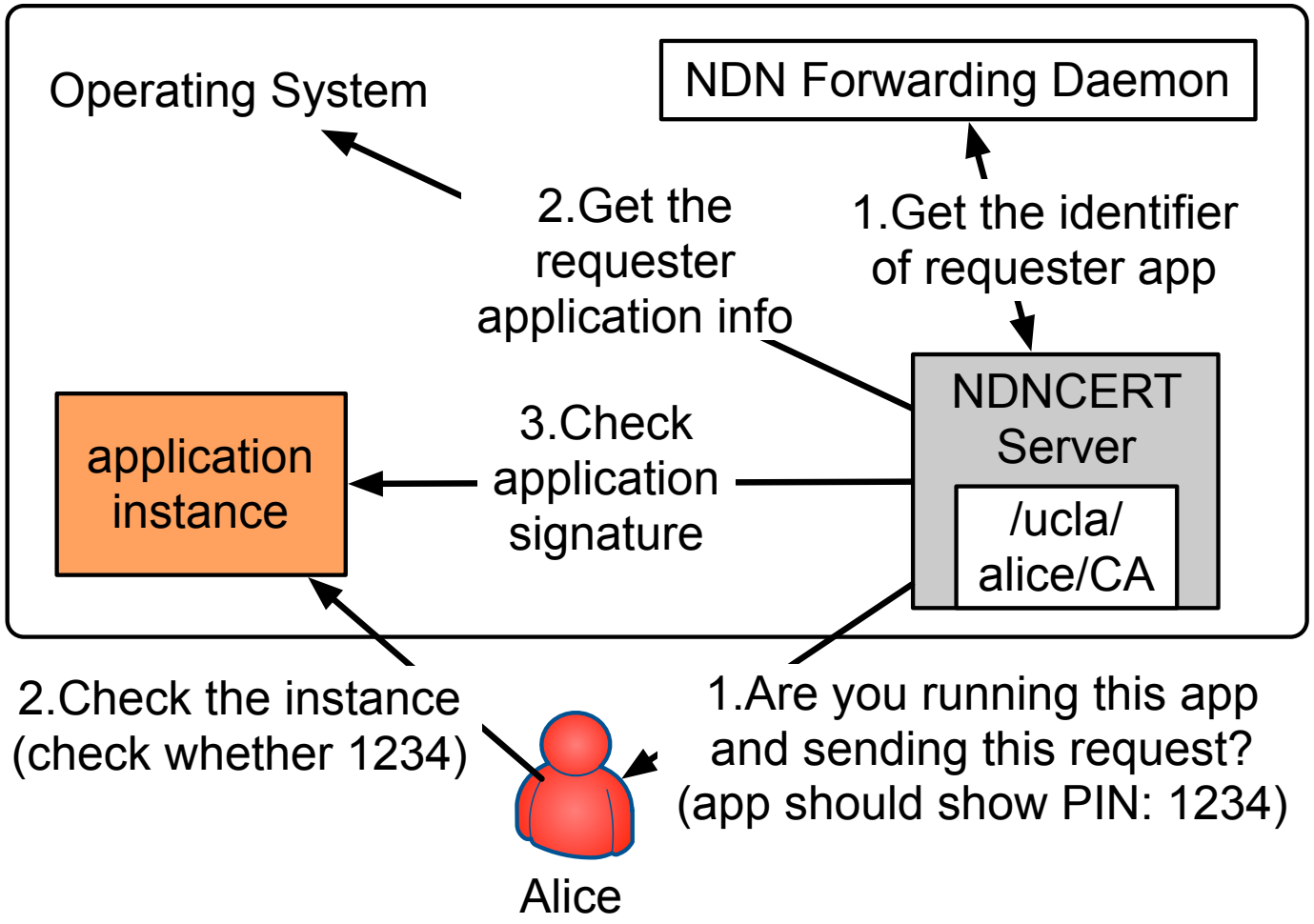
NDNCERT in Action (Initiate Request)



NDNCERT in Action (Prove "Yourself" and Get Cert)



Intra-Node (App) Certificates



- Make sure request came from the right application instance
  - App was developed by the trusted developer and the code could has not been tampered with
  - App instance is run by trusted user

NDNCERT API Highlights

```
client.sendProbe(ca, "...",
                 [] (...) {
                   sendNew(ca, certIdentity,
                           [] (...) {
                             sendSelect(ca, certIdentity,
                                        [] (...) {
                                          sendValidate(ca, ...,
                                                       [] (...) {
                                                         requestStatus / requestDownload
                                                       });
                                        }
                                        ...);
                           },
                           ...);
                 }
                 , ...);
```

NDNCERT Next Steps

- Simpler API to be used in applications (for intra-node certs)
- Integration with NDN Control Center
- Integration with NDN Android

Common Name Library - Motivation

- Explore a *collection-oriented* API for NDN application developers
- Provide tools for working with namespaces as they represent collections, in an *information-focused* rather than communication-oriented way
- Assume asynchronous network operations will be used to sync the namespace and consume/publish objects in the collection
- Make progress towards NDN as a middleware-replacement in terms of high-level, application-facing features, but try to stay as general as possible
- Refine the set of tools for working with common naming patterns
- Provide a lightweight way to integrate various:
  - Sync mechanisms (i.e., ChronoSync, Psync),
  - Data access patterns (i.e., Consumer/Producer API, RDR)
  - Publishing models (i.e., C/P API, in-memory content cache),
  - Namespace queries / pattern matching (i.e., regexp, wildcard components),
  - Triggered data generation (supporting security)

Common Name Library

- Library enabling applications to work with hierarchical, named data collections
  - Namespace object (root and child nodes)
  - Application interacts with a Namespace node (attach handlers, receive notifications)

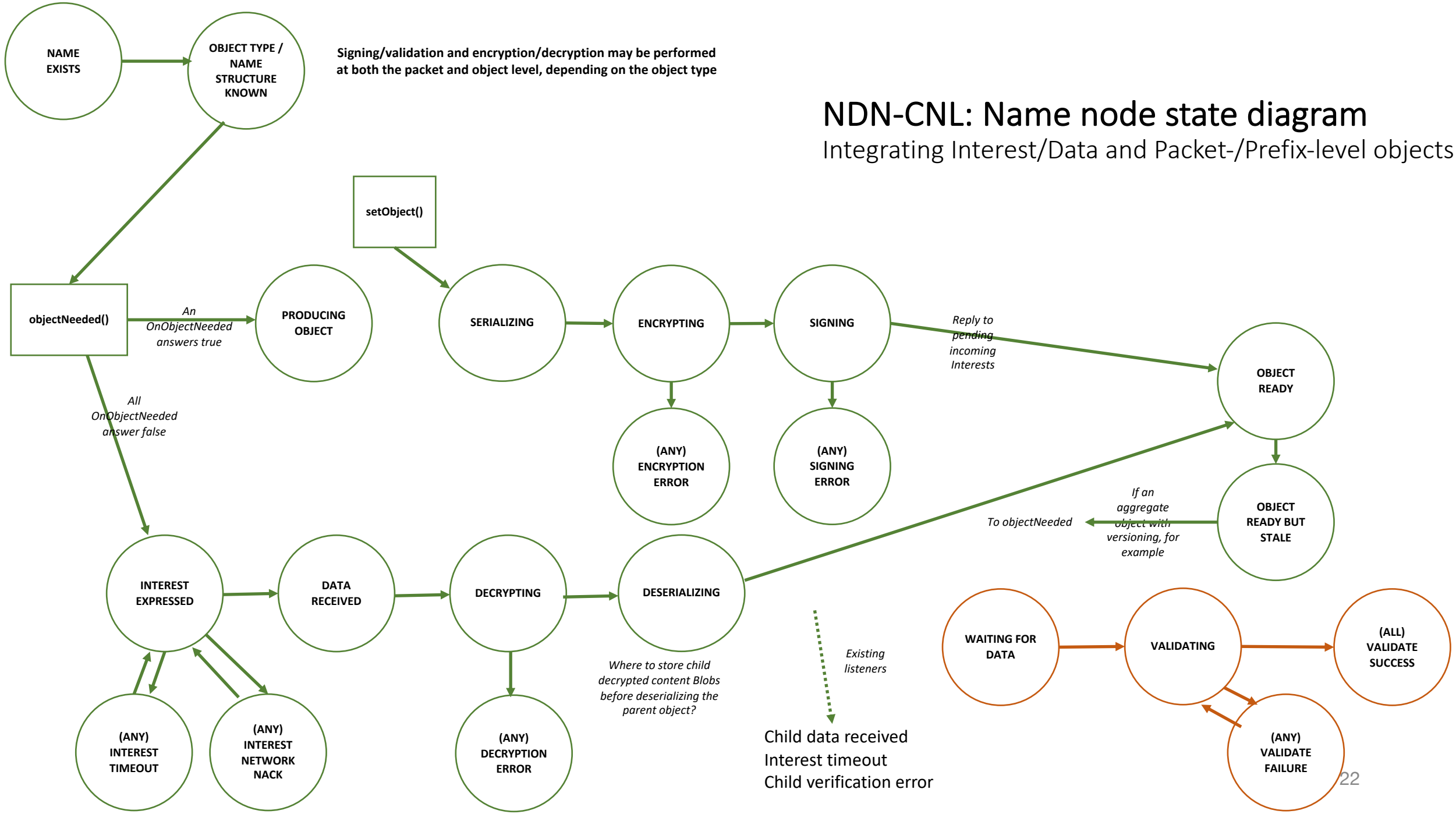
```
from pyndn import Face
from pycnl import Namespace, SegmentedObjectHandler

def main():
    face = Face("memoria.ndn.ucla.edu")
    page = Namespace("/ndn/edu/ucla/remap/demo/ndn-js-test/named-data.net/project/ndn-ar2011.html/%FDT%F7n%9E")
    page.setFace(face)

    def onSegmentedObject(handler, obj):
        print("Got segmented object")

page.setHandler(SegmentedObjectHandler(onSegmentedObject)).objectNeeded()
```

CNL - Unified publisher/consumer

- objectNeeded() – From application (producer) or network (consumer)
- Producer
  - CNL receives Interest, adds to PIT, calls OnObjectNeeded (if not already in cache).
  - Handler's OnObjectNeeded answers True.
  - CNL waits for application to produce data asynchronously.
  - Application calls setObject().
  - CNL does serialize/encrypt/sign and satisfies PIT.
- Consumer
  - Application calls OnObjectNeeded for a Namespace node. (All handlers answer False.)
  - CNL does Face.expressInterest and waits for Data.
  - CNL receives Data, does verify/decrypt/deserialize and OnStateChanged(OBJECT_READY)

NDN-CNL: Name node state diagram

Integrating Interest/Data and Packet-/Prefix-level objects



CNL – Next steps

- High-performance persistent storage
- Psync for name space synchronization
  - Efficient use of Bloom filters to resolve differences in sets of names
  - Robust to network partitions
  - Builds on ChronoSync research
- More applications
  - Currently used in augmented reality mobile client application
- More protocols
  - Perhaps RDR

How to learn more

- NAC
  - https://github.com/named-data/name-based-access-control
- NDNCERT
  - https://github.com/named-data/ndncert
- Common Name Library
  - C++: https://github.com/named-data/cnl-cpp
  - C# (for Unity): https://github.com/named-data/cnl-dot-net
  - Python: https://github.com/named-data/PyCNL