# Computing the Distribution of Computations for Named Function Networking Using Name Based Routing

Masterthesis

Christopher Scherb

Natural Science Faculty of the University of Basel

Department of Mathematics and Computer Science

Computer Network Research Group

cn.cs.unibas.ch

# Current internet situation

- TCP/IP standard in the current internet
  - Does not fit requirements for content distribution
  - Large content files

- Information Centric Networks (ICN)
  - Request data with name
  - Focus on data instead of connections

# Motivation

- Internet does not only consist of static data
  - Web services
  - Images/Video in different formats etc.

- Cloud networks and distributed computing
  - Big data processing

- ICN deliver only static data

# Idea

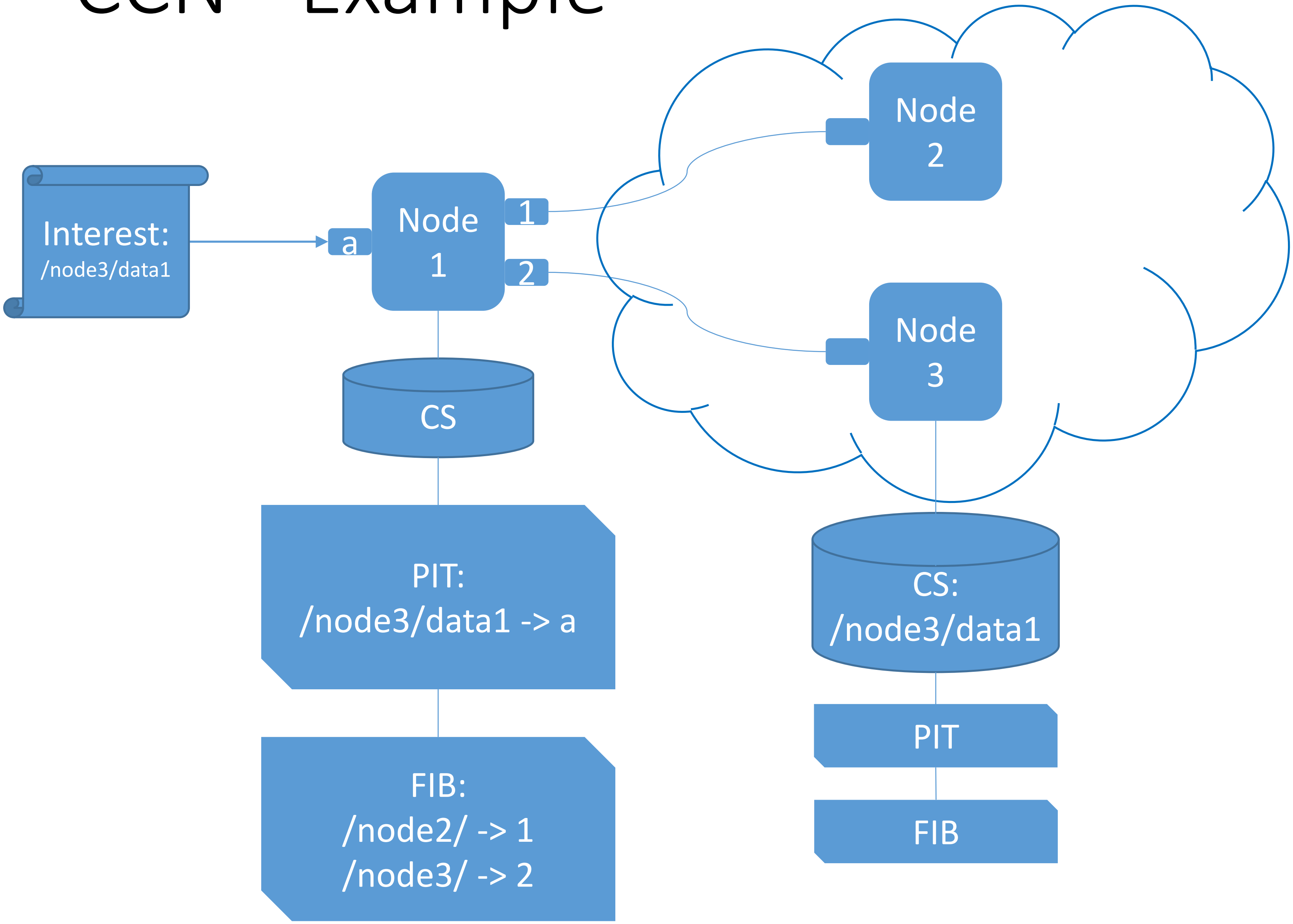
- Extended ICN:
  - Deliver results
  - Fetching data is often only a first step
  - Data should be processed
  - Fit requirement of many users
  - Reduce network traffic

- Named Function Network (NFN)

# Content Centric Networking

- Developed by PARC
- Focus on data, not on connections
    - Request named data: Interest
      Reply: content object
    - Caching of data next to users possible
- Name based routing
    - Hierarchical name structure
- Important data structures:
    - FIB – forwarding information
    - PIT – pending interests
    - CS – cache

# CCN - Example


Interest:
/node3/data1
a
Node
1
1
2
CS
PIT:
/node3/data1 -> a
FIB:
/node2/ -> 1
/node3/ -> 2
Node
2
Node
3
CS:
/node3/data1
PIT
FIB

# Named Function Networking

- User defines how data should be delivered
- Network decides how to
    - Distribute computations
    - Deliver a result form the network
    - Use cached results to avoid recomputations
- Represent programs in CCN-names
    - Encoded in $\lambda$-calculus

# $\lambda$-Calculus

- Formal mathematical language
- Express computations based on
  - Variables: $x$
  - Abstractions: $\lambda\, x.\, M$
  - Applications: $M\ N$

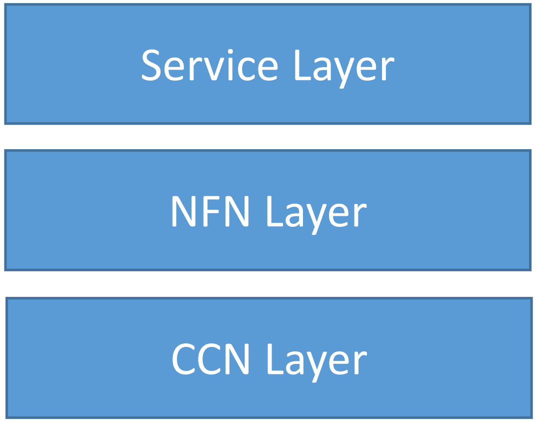
- Basis for functional programming languages
- Pure $\lambda$-calculus is Turing complete
  - Numbers/Operations can be represented in $\lambda$-expressions
- Executed by reducing it in an abstract machine

→ CCN-nodes are extended with an abstract machine

# Native Functions

- Support for build-in/native functions in the abstract machine
    - Programmability
    - Performance
- Native functions are stored in content objects
    - Can be transferred over the network
    - Can be executed by every node
- Pinned functions are only available on one node
    - E.g. for security or copyright reasons
    - Data have to be transferred to the function
- Black-box execution for the abstract machine
    - Appears as one machine step

# Three Layers



Service Layer

Execute complex operations/native functions,
data manipulation

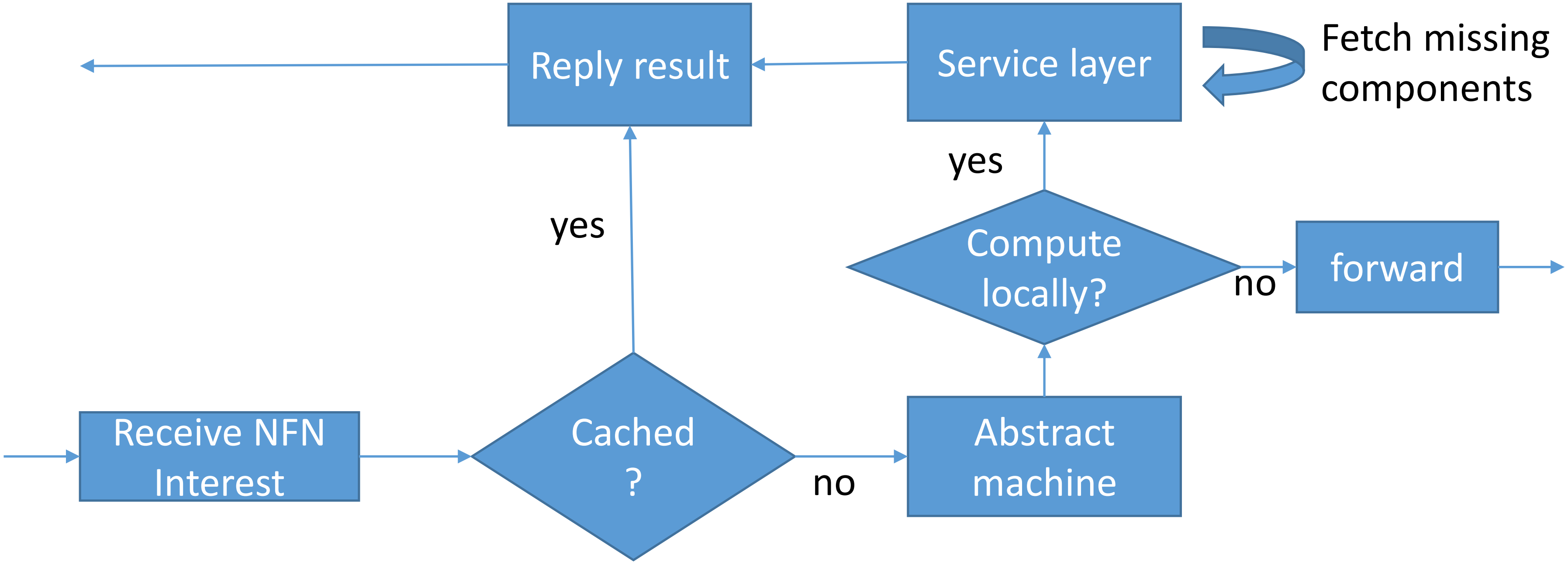
NFN Layer

Compositing and distributing computations,
name manipulation

CCN Layer

Forwarding Packets,
no name manipulation

# NFN workflow



# Name Encoding

- Interests: `call /func/wc /docs/doc1`
    - No routable name
    - Cannot be routed by a CCN node
- Apply an abstraction:
    - ($\lambda$ x.call /func/wc x) `/docs/doc1`
    - Still not routable
- Change order:
    - `/docs/doc1` ($\lambda$ x.call /func/wc x)
    - Routable with longest prefix matching
    - Ignore the computation and use the name
- Only the node that receives a non routable interest changes names
    - Organizer node
    - Responsible for finishing a computation

# NFN-Layer Strategy

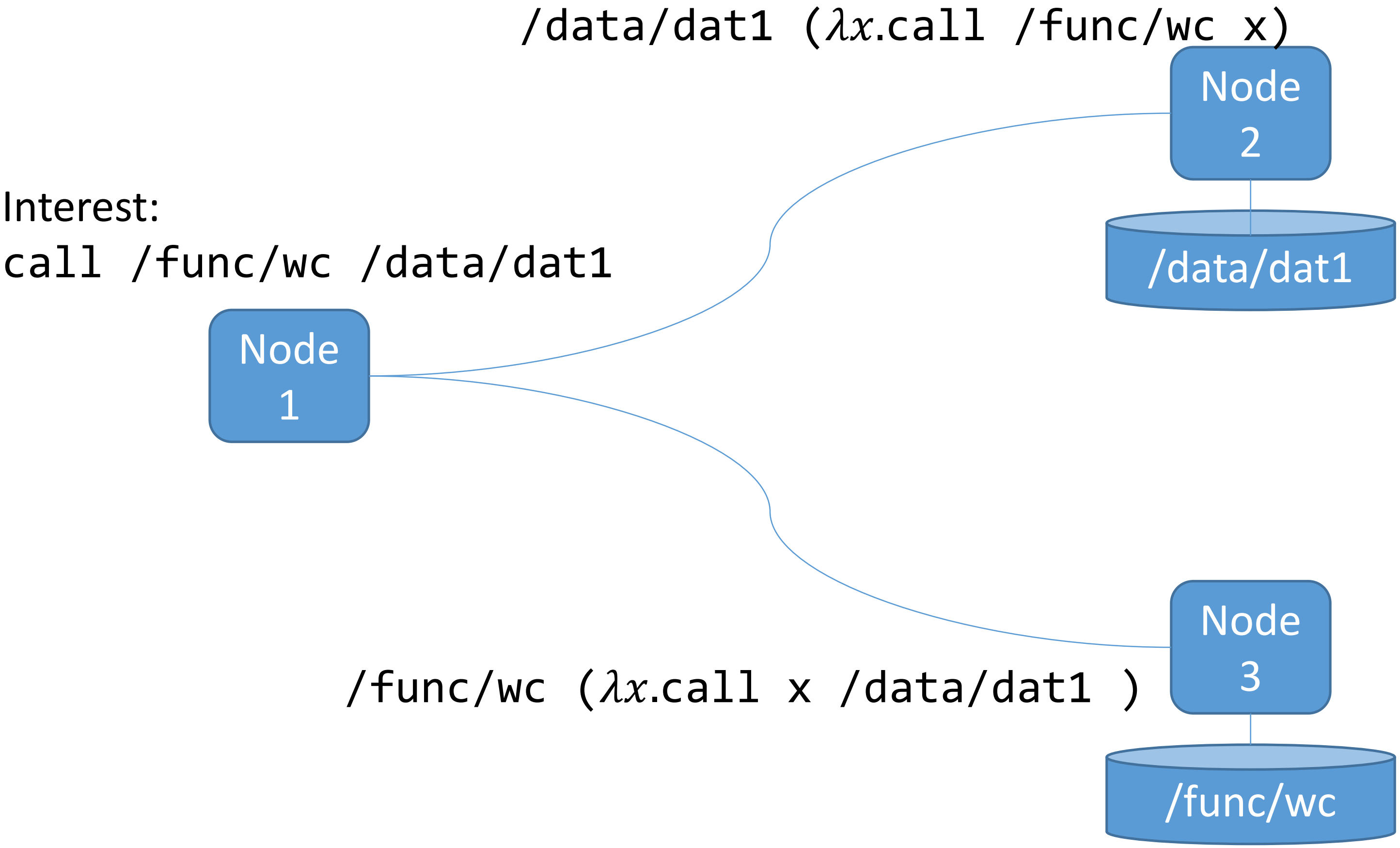
- Multiple names in an expression
  - Which one should be used?

1. Prepend input data name first
   - Route to a node with the input data
2. If fail prepend function name
   - Route to a node with the function

- Computation is started on a node that has the prepended name locally available

- Why can a computation fail?
  - Pinned function
  - Data on CCN-only node

# Example



`/data/dat1 (`$\lambda x$`.call /func/wc x)`

Interest:
`call /func/wc /data/dat1`

`/func/wc (`$\lambda x$`.call x /data/dat1 )`

# Other Strategies

- Different strategies can be employed
  - Current: last name first
- Local strategies:
  - First name first, last name last
  - Most explicit name first (most components)
- Fetch additional information from the network
  - Name with largest content first
  - Name which issues smallest network traffic first

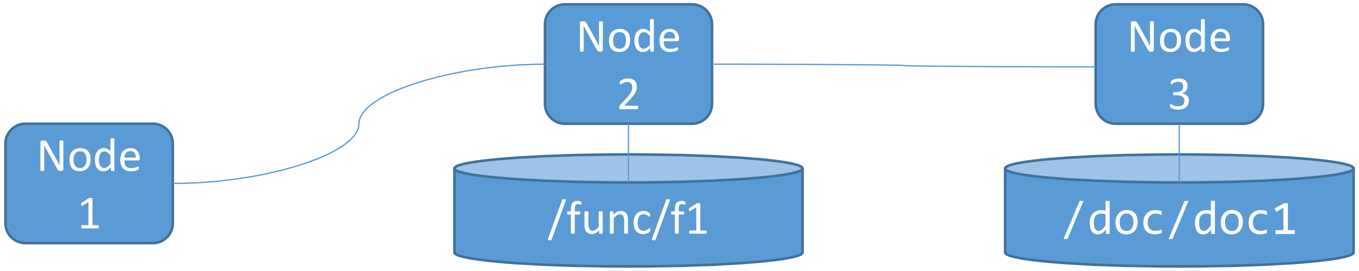
# Caching

- CCN caches data to reduce network traffic
- NFN caches results to reduce computational load

- Only cache results of service layer operations
    - Computing an abstract machine step is faster than fetching it
    - Time to request and to transfer result must be smaller than the computation time
    - Otherwise the computation would not benefit

# Caching Example

- `call /func/f1 (call /func/f2 /doc/doc1)`
- First compute inner result on a node with /doc/doc1
  - `call /func/f2 /doc/doc1`
- Result will be cached
- Compute entire result on node with /func/f1
- If `call /func/f2 /doc/doc1` is cached use it



# New challenges

- Working NFN system available
- Still problems
- Analysis of some problems and solution approaches

# Load Balancing

- Usually nodes next to the user are organizer nodes
  - Many users and many requests can overload organizer nodes
  - With computations DoS attacks become easy
- Accept only computations if there are free resources
- If not choose a neighboured node
  - Forward the interest without changing the name
  - Nodes deep in the network can become organizer nodes

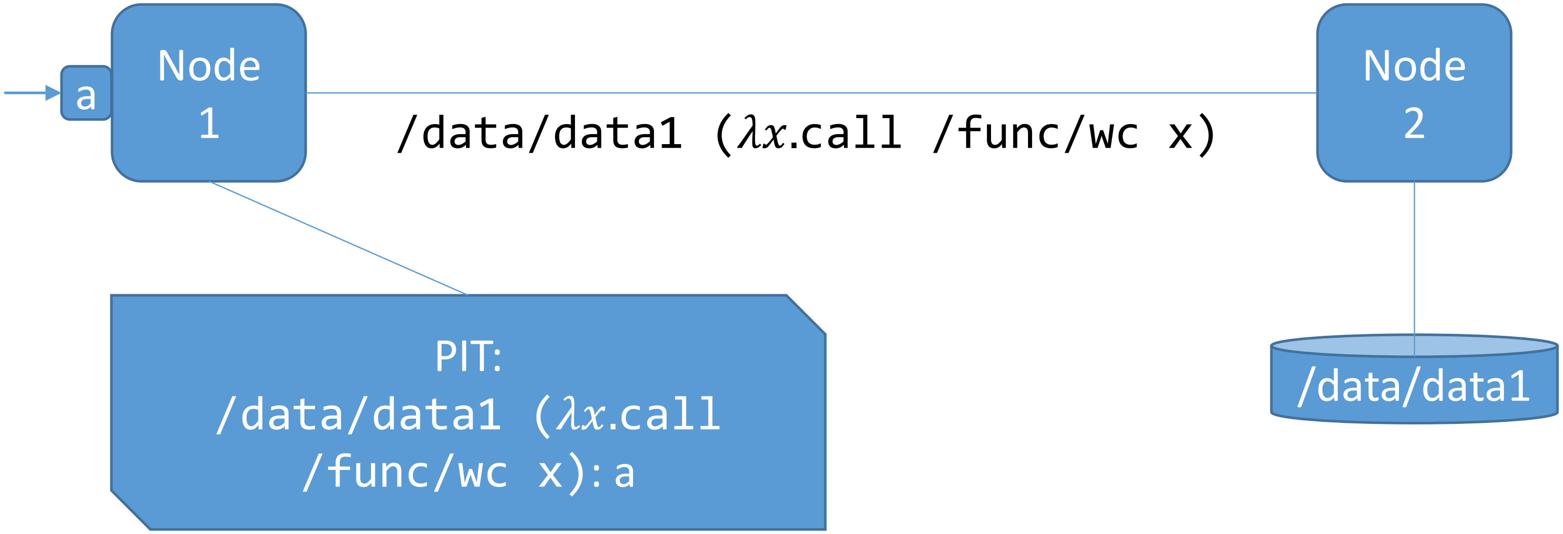
# Long running computation

- Interests in the CCN-PIT timeout
    - What if a computation requires more time?
- Node cannot distinguish between failed and timeout
- First request explicit if
    - A computation can be performed
    - How long it would take?

→Thunks

# Thunk Example

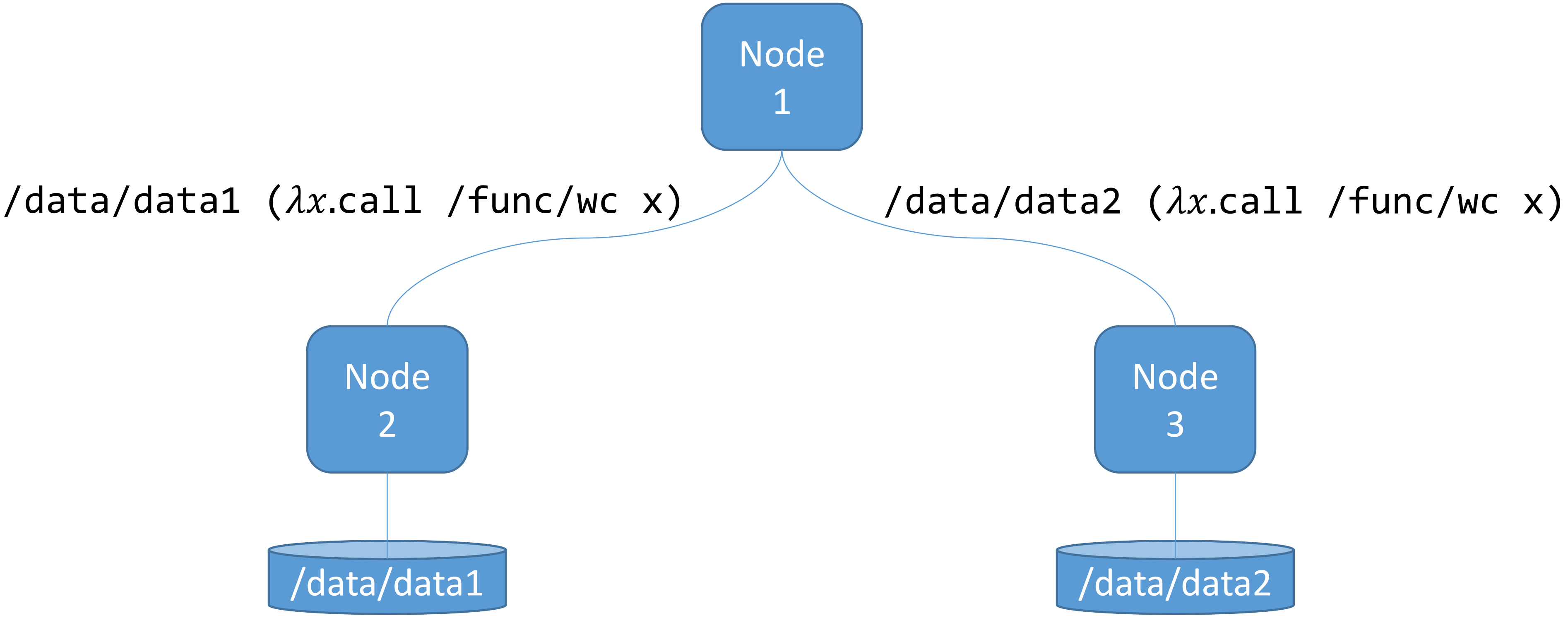
Interest:
`call /func/wc /data/data1`


a
Node 1
/data/data1 (λx.call /func/wc x)
Node 2
PIT:
/data/data1 (λx.call /func/wc x): a
/data/data1

# Thunk Parallel Example

Interest:

`add (call /func/wc /data/data1) (call /func/wc /data/data2)`



Node 1

/data/data1 ($\lambda x$.call /func/wc x)

/data/data2 ($\lambda x$.call /func/wc x)

Node 2

Node 3

/data/data1

/data/data2

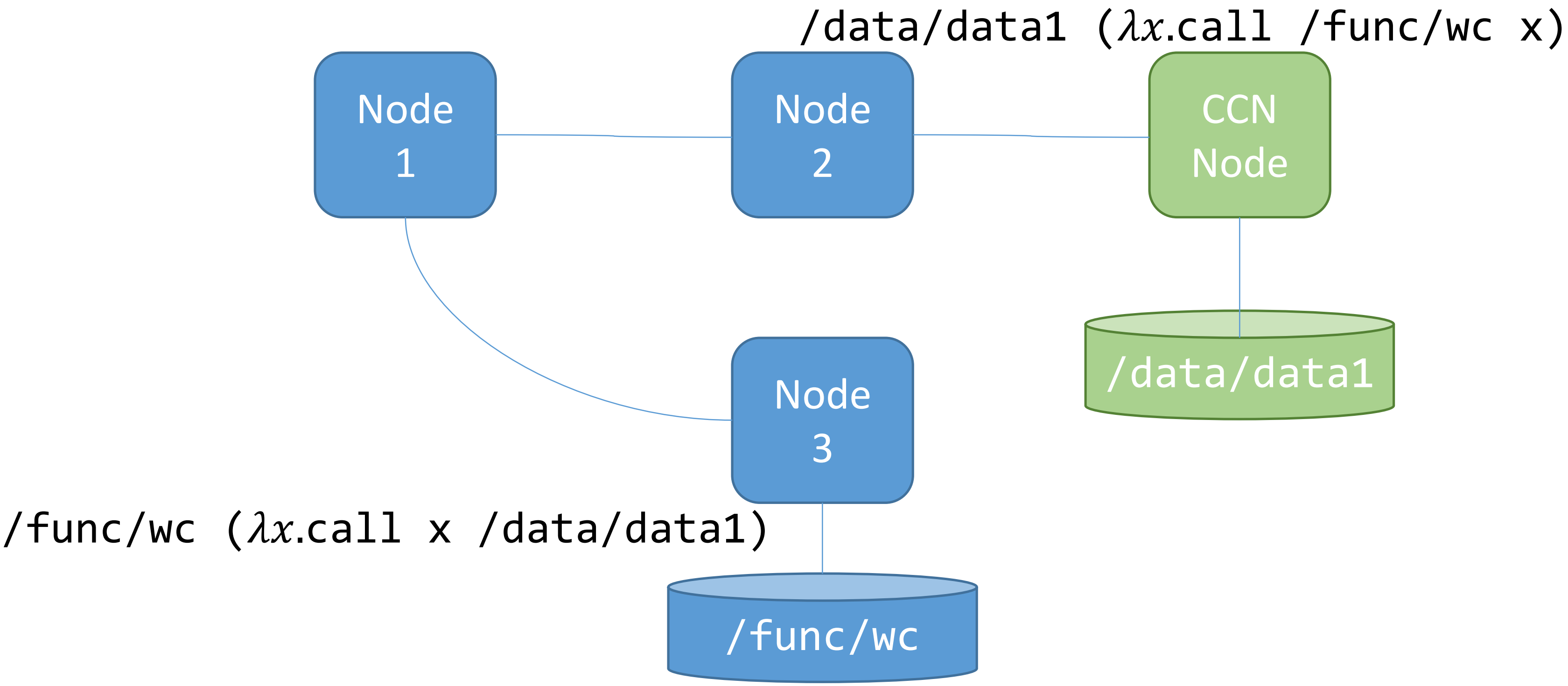
# Avoid timeouts

- Inform the previous node that a computation cannot be finished
- Reply with a negative acknowledgement (NACK) message
- Avoid timeouts
  - Organizer node not involved in networks with CCN-only nodes
  - Faster reaction time
  - Computation closer to a CCN node
- NACKs for CCN and NFN

# Nack Example

Interest:

`call /func/wc /data/data1`



/data/data1 ($\lambda x$.call /func/wc x)

/func/wc ($\lambda x$.call x /data/data1)

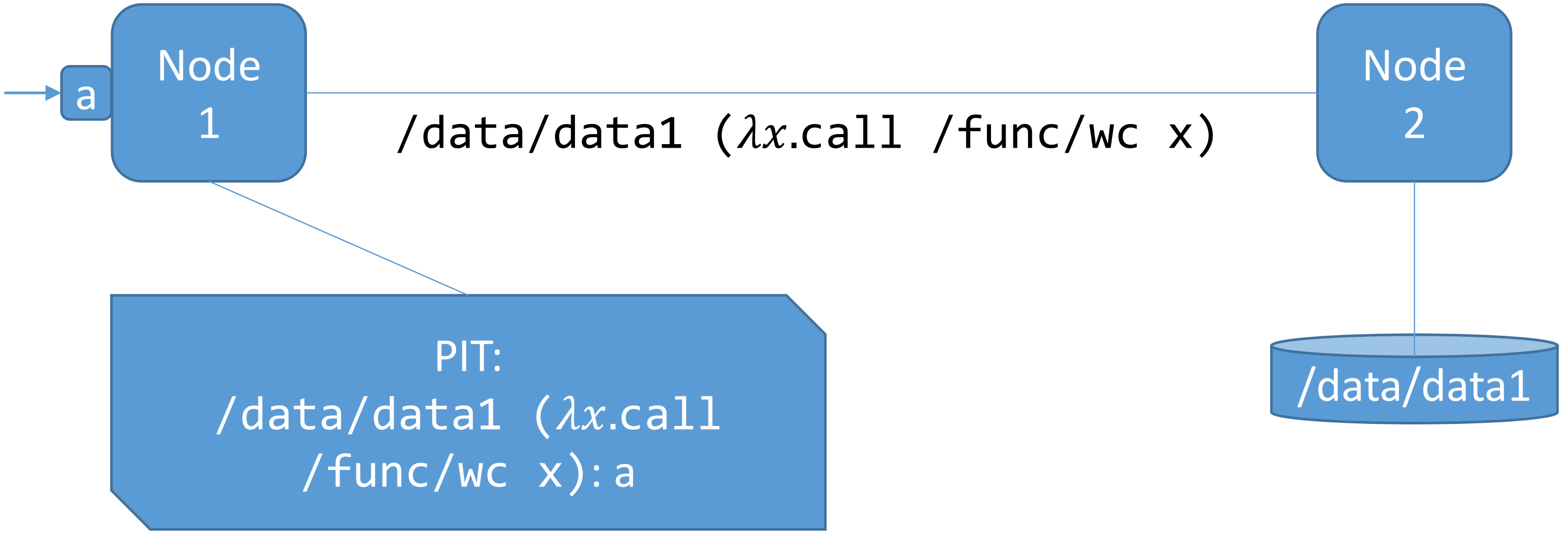
# Discussion

- Signing of results
    - Sign by the data source or the computation node?
- NFN data manipulation
    - In $\lambda$-calculus
    - Head/tail function for iterating files
- Improving thunks
- Different NACK types

# Improving thunks

Interest:

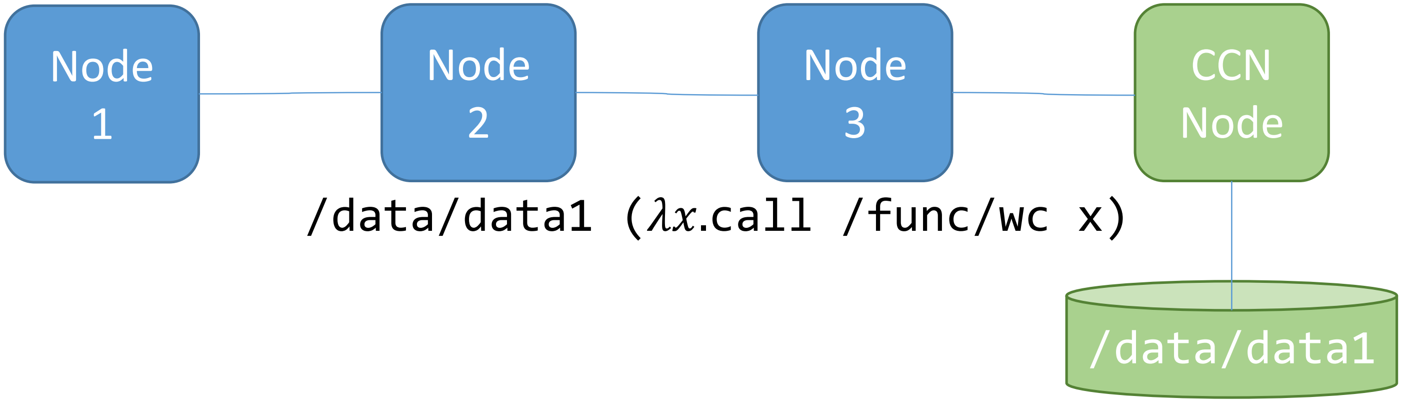
`call /func/wc /data/data1`


a
Node 1
/data/data1 (λx.call /func/wc x)
Node 2
PIT:
/data/data1 (λx.call /func/wc x): a
/data/data1

# Improving NACKs

Interest:

`call /func/wc /data/data1`



Node 1

Node 2

Node 3

CCN Node

/data/data1 ($\lambda x$.call /func/wc x)

/data/data1

# Summary

- CCN solves problems with content distribution in current internet
- CCN only deliver static data
- What about web services etc.?
- CCN extend to NFN
  - User define result to be delivered
  - Abstract machine in CCN-nodes
  - Distribute computations over a network
  - Optimize execution location

# Conclusion

- Working prototype
- Address problems to enable the system for general purpose computing
  - Thunks for long running and parallel computations
  - NACKs for faster reaction time
- To perform computations
  - Similar to Internet and web services
  - CCN is extended with NFN
    - But perform arbitrary computations

Thank you for your attention.