Pastry:
An example of a distributed flat naming system

Konrad Iwanicki
University of Warsaw

Supplement for Topic 05: Naming
Distributed Systems Course
University of Warsaw

Based on: A. Rowstron and P. Druschel, **“Pastry: Scalable, distributed object location and touting for large peer-to-peer systems,”** in *Middleware 2000: Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms*, Heidelberg, Germany, November 2001, pp. 329-350.

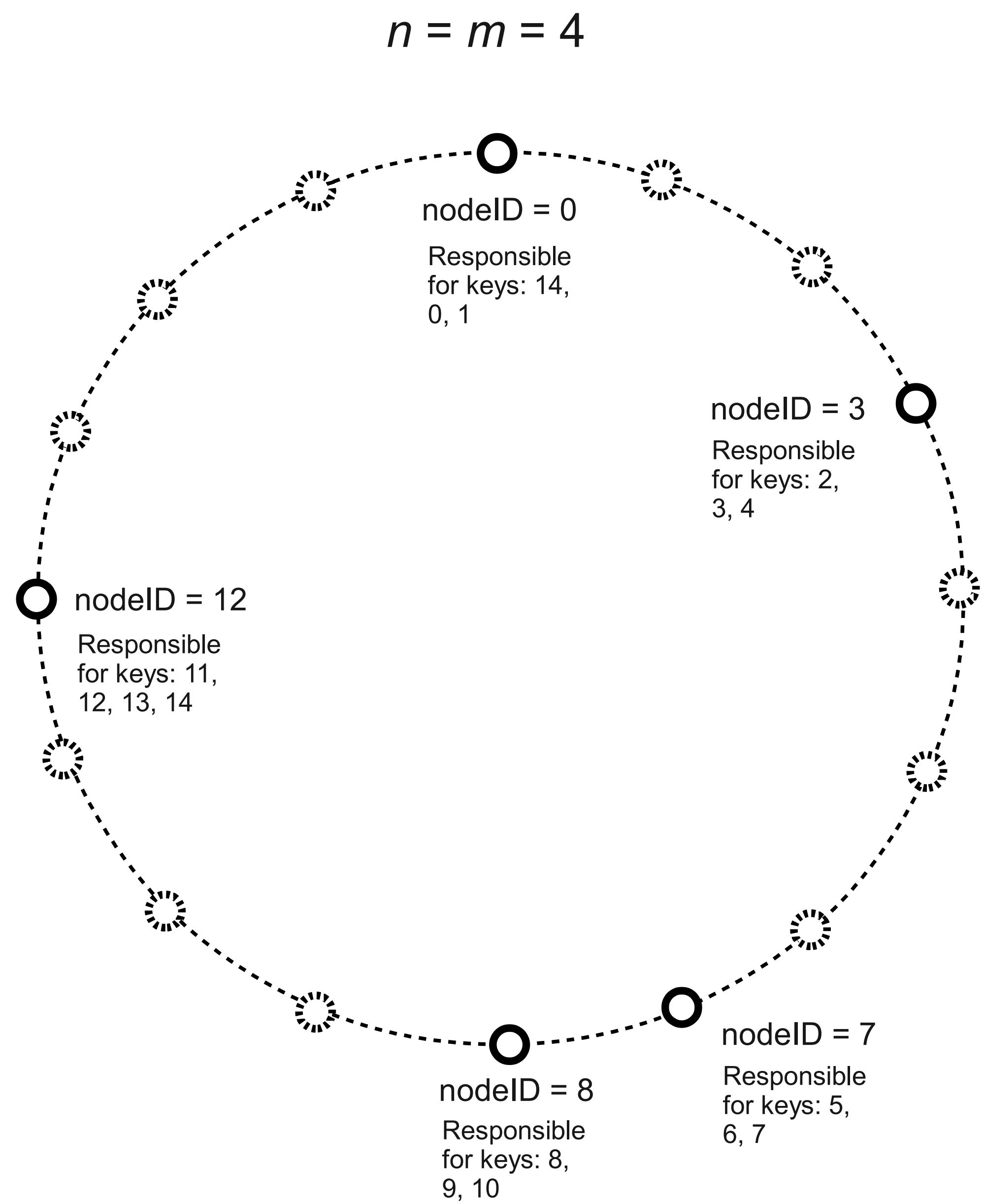
Introduction

- Pastry provides a mechanism for resolving flat entity names into entity addresses:
  - Each entity (e.g., file, object, service) is given a flat *m*-bit identifier – a key.
  - Each entity is hosted by some node.
  - Given
    - a key of an entity and
    - a transport-layer address of **any** node,

  Pastry locates the transport-layer address of a node hosting the entity corresponding to the key.

Overview

- Each node is assigned a random, unique *n*-bit identifier: nodeID ($n \leq m$, typically $n = m = 128$).
- NodeIDs constitute a numeric space ranging from 0 to $2^n - 1$.
- A node hosts entities whose keys are numerically closest to its nodeID.



Overview

- **QUESTION:** Why is it important that node identifiers be random?

Overview

- **QUESTION:** Why is it important that node identifiers be random?
- Assuming that keys are also uniformly random (e.g., generated by a cryptographic hash function), the entities will be well-balanced between nodes.

Overview

- **QUESTION:** What is the advantage of mapping each keys to the numerically closest nodeID when the population of nodes changes?

Overview

- **QUESTION:** What is the advantage of mapping each keys to the numerically closest nodeID when the population of nodes changes?
- Entity transfers are local: entities only from the two nodes with numerically closest nodeIDs are potentially affected.

Overview

- Pastry nodes form an *overlay network*, in which each node has links to selected other nodes.
- Those links are used to route a lookup message for a key from a source node to a destination node that hosts the entity with the key.
  - This is overlay routing => at the application layer.
- Think of keys and nodeIDs as numbers with base 2^b digits, where b is a configuration parameter (typically b = 4).
- Pastry can route a lookup message within $\lceil \log_{2^b} N \rceil$ hops over the overlay links, (N = the total number of nodes).
- To this end, each node maintains a local state.

Node state

- Leaf set
- Routing table
- Neighborhood set

Leaf set

- Contains entries for *L/2* smaller and *L*/2 larger numerically closest active nodeIDs.
  - *L* is a configuration parameter (typically 16 or 32)
- An entry for a nodeID consists of, among others, the transport-layer address of the node with the nodeID.

Leaf set for nodeID = 10233102
(b = 2, n = 16, #digits = n/b = 8, L = 8)

SMALLER		LARGER	
10233033	10233021	10233120	10233122
10233001	10233000	10233230	10233232

Routing

- If we used just leaf sets, routing could work as follows:
  - forward the message to the node from the leaf set numerically closest to the key.

Routing

- If we used just leaf sets, routing could work as follows:
  - forward the message to the node from the leaf set numerically closest to the key.
- **QUESTION:** How many overlay hops would we need?

Routing

- If we used just leaf sets, routing could work as follows:
  - forward the message to the node from the leaf set numerically closest to the key.
- **QUESTION:** How many overlay hops would we need?
- (N / 2) / (L / 2) = N / L
  - With $N = 2^{128}$ and L = 32 this is poor => such routing does not scale in terms of the system size.

Routing table

- Organized into rows:
  - $2^b - 1$ entries per row.
- Entries in row *i*, each refer to a node:
  - whose nodeID equals the present node's nodeID in the first *i* digits
  - And differs from the present node's nodeID in the *i*+1-st digit

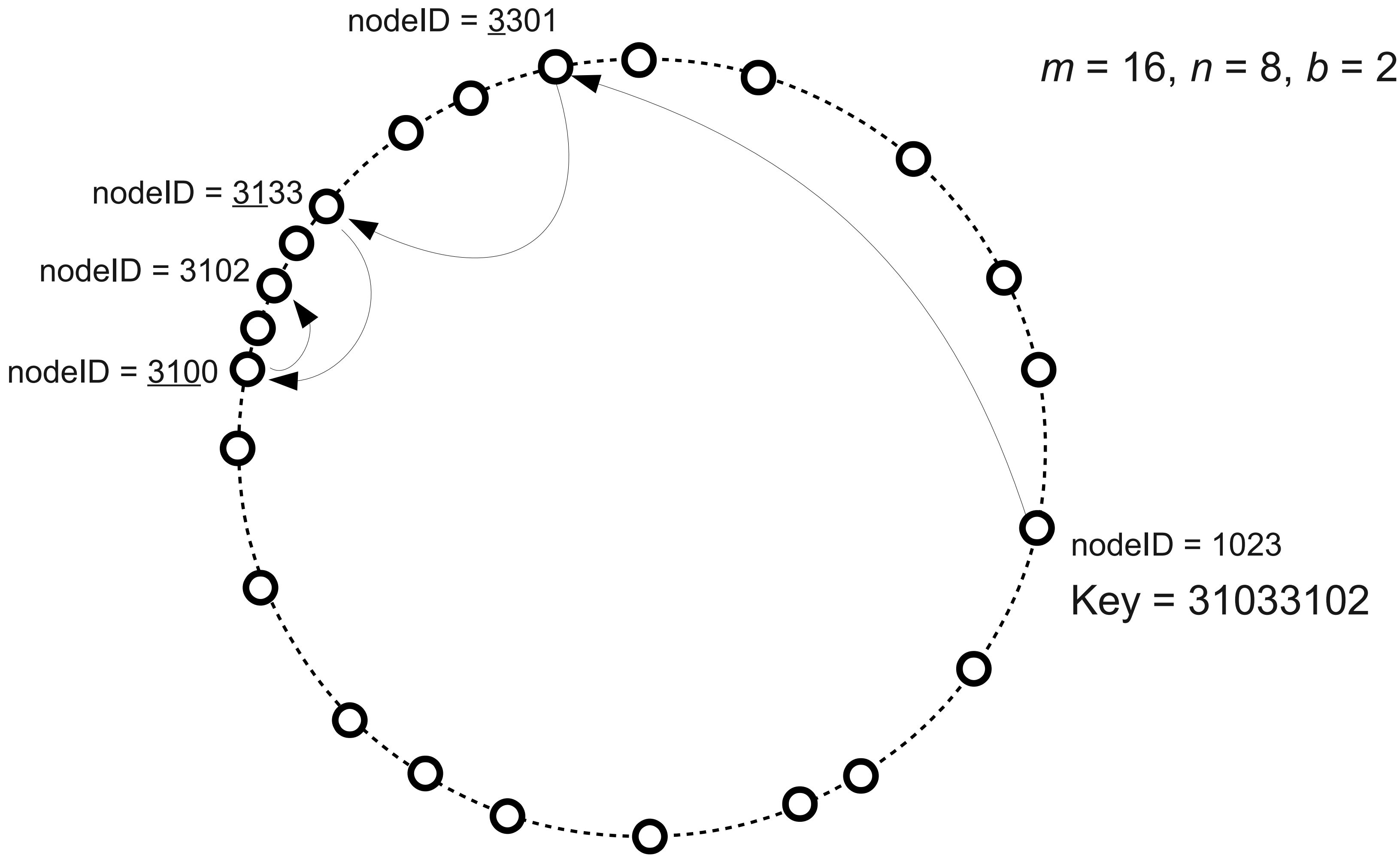
Routing table for nodeID = 10233102
(*b* = 2, *n* = 16, #digits = *n*/*b* = 8)

Row	0	1	2	3
0	-**0**-2212102	1	-**2**-2301203	-**3**-1203102
1	0	1-**1**-301233	1-**2**-321333	1-**3**-120123
2	10-**0**-31203	10-**1**-32102	2	10-**3**-22312
3	102-**0**-0230	102-**1**-1231	102-**2**-0001	3
4	1023-**0**-011	1023-**1**-301	1023-**2**-022	3
5	10233-**0**-01	1	10233-**2**-31	
6	0		102331-**2**-1	
7			2	

Routing

- If the key in a lookup message is within the range of the leaf set,
  - forward the message to the node from the leaf set numerically closest to the key.
- Else if there exists in the routing table an entry whose nodeID shares one more digit with the key than the nodeID of the present node,
  - forward the message to the node corresponding to the entry.
- Else we have to decide if the present node should accept the message:
  - If some entry in the leaf set is numerically closer to the key than the present node:
    - Forward the message to the node corresponding to the numerically closer entry from the leaf set.
  - Else:
    - Accept the message as the destination node responsible for the key.

Routing


nodeID = 3301
m = 16, n = 8, b = 2
nodeID = 3133
nodeID = 3102
nodeID = 3100
nodeID = 1023
Key = 31033102

Routing

- If a key falls within the leaf set, just 1 hop is needed.
- Otherwise, at each hop, **at least** one base 2^b digit is resolved.
  - At each hop, the number of candidate nodes that can potentially host the key is thus reduced by a factor of 2^b.
  - In the end, the number of candidate nodes has to be narrowed down to 1.
  - The number of hops, *h*, that is necessary thus satisfies the equation:
    - $N / (2^b)^h \approx 1 \Rightarrow N \approx (2^b)^h \Rightarrow h \approx \log_{2^b} N$.
- Such routing scales well wrt. the system size, *N*.

Routing

- **QUESTION:** With such efficient routing, is there any sense to have the leaf set bigger than just two entries, that is, to have $L / 2 > 1$?

Routing

- **QUESTION:** With such efficient routing, is there any sense to have the leaf set bigger than just two entries, that is, to have $L / 2 > 1$?
- $L / 2 > 1$ is necessary for fault tolerance.
  - The links corresponding to the leaf set build the ring:
    - If $L / 2 = 1$ then each node has a pointer to its ring successor and predecessor.
    - If $L / 2 = k$ then each node has a pointer to its k ring successors and k predecessors.
  - If k consecutive nodes fail concurrently, the ring is broken.
  - It thus makes sense to make k large: $k - 1$ concurrent node failures can be tolerated.

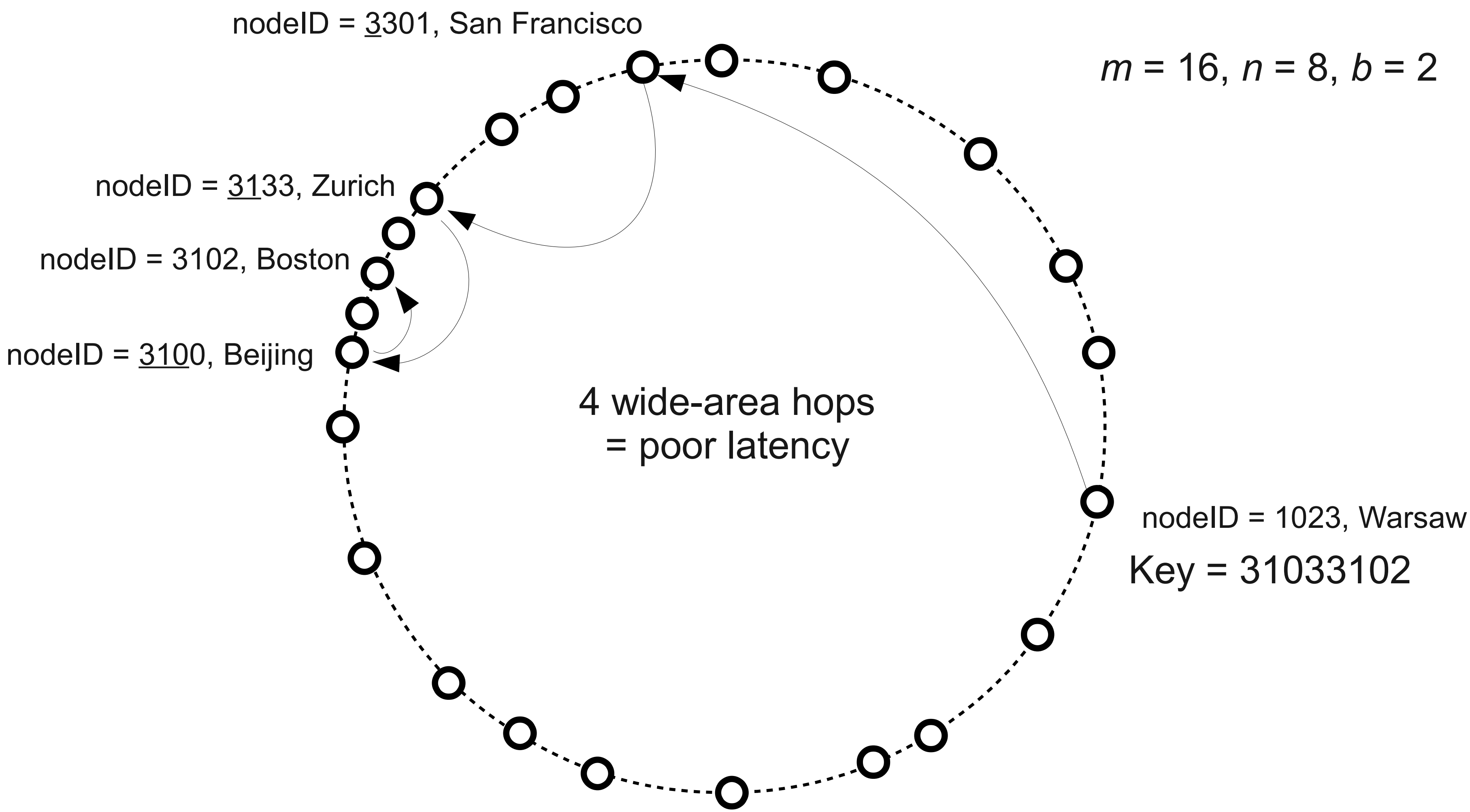
Routing

- Pastry is scalable in network size.
- **QUESTION:** What about geographic scalability?

Routing

- Pastry is scalable in network size.
- **QUESTION:** What about geographic scalability?
- The design presented so far does not scale well geographically.

Routing


nodeID = 3301, San Francisco
m = 16, n = 8, b = 2
nodeID = 3133, Zurich
nodeID = 3102, Boston
nodeID = 3100, Beijing
4 wide-area hops
= poor latency
nodeID = 1023, Warsaw
Key = 31033102

Routing

- A possible solution:
  - Assign nodeIDs in a geographically aware manner: nodes close on the ring are also close geographically.

Routing

- A possible solution:
  - Assign nodeIDs in a geographically aware manner: nodes close on the ring are also close geographically.
- **QUESTION:** Drawbacks?

Routing

- A possible solution:
  - Assign nodeIDs in a geographically aware manner: nodes close on the ring are also close geographically.
- **QUESTION:** Drawbacks?
  - Mapping a one-dimensional space to the Internet is far from trivial.
  - Correlated failures:
    - When an enterprise network goes down, many consecutive nodes go down.
    - The ring can break.

Routing

- Pastry thus uses something else:
  - *proximity neighbor selection*.
- The third element of a node's state – the neighbor set – contains *M* entries for nodes "close" to the present node (typically *M* = 32):
  - "Close" in some proximity metric (e.g., latency), **not** in the nodeID space.
- This set is used to construct a routing table that has good locality.

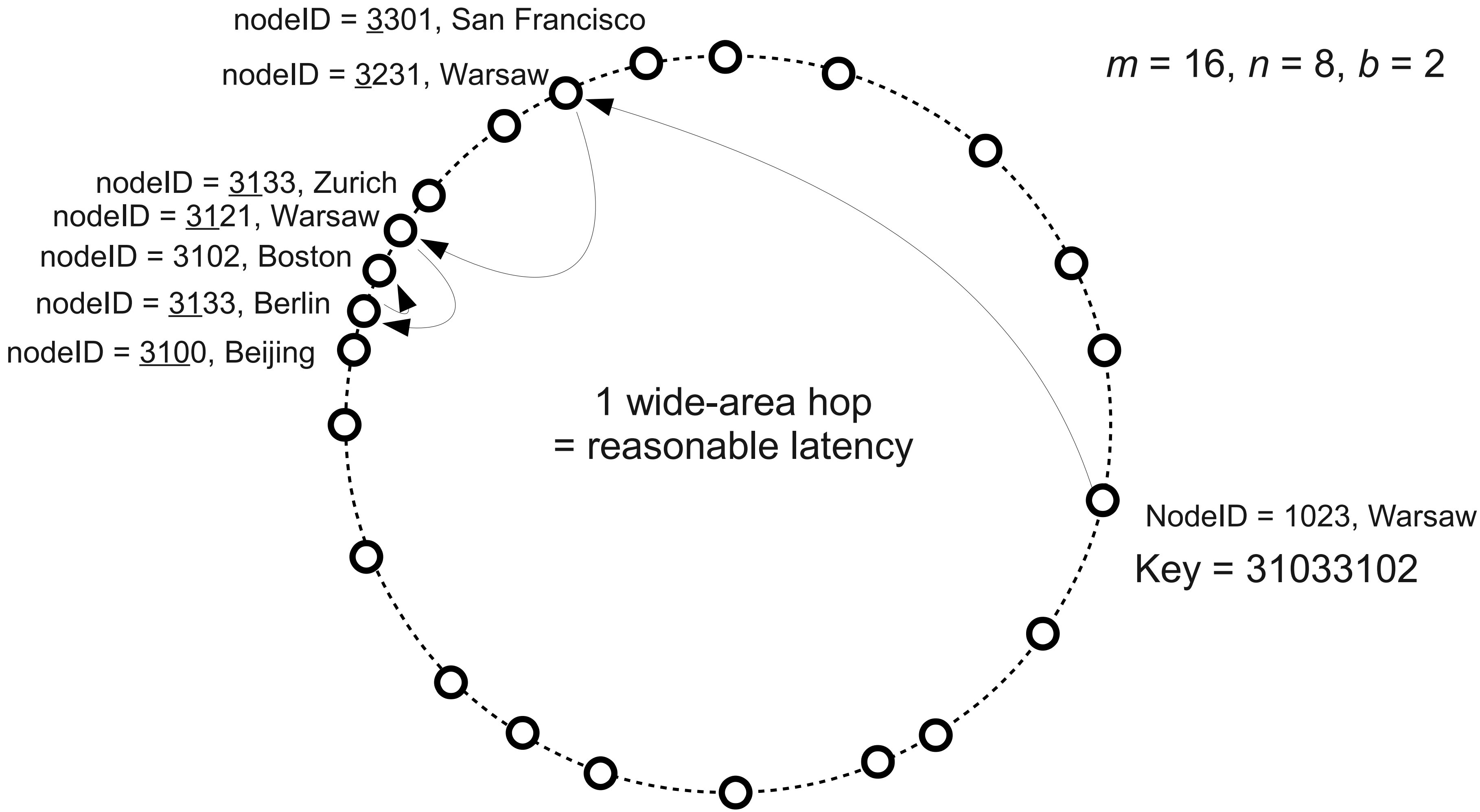
Routing

- Idea: There is a lot of choice in selecting entries for the routing table.
- An entry for row 0 can be selected from N / 2^b nodes.
  - A a close node with a matching nodeID is likely to exist.
- An entry for row *r* can be selected from N / ($2^{b(r + 1)}$) nodes.
  - In rows apart from the last few ones, close nodes with matching nodeIDs are likely to exist.
- Conclusion: for the links, we can select entries close in the proximity metric.

Row	0	1	2	3
0	-**0**-???????	1	-**2**-???????	-**3**-???????
1	0	1-**1**-??????	1-**2**-??????	1-**3**-??????
2	10-**0**-?????	10-**1**-?????	2	10-**3**-?????
3	102-**0**-????	102-**1**-????	102-**2**-????	3
4	1023-**0**-???	1023-**1**-???	1023-**2**-???	3
5	10233-**0**-??	1	10233-**2**-??	
6	0		102331-**2**-?	
7			2	

Details in the paper.

Routing


nodeID = 3301, San Francisco
nodeID = 3231, Warsaw
nodeID = 3133, Zurich
nodeID = 3121, Warsaw
nodeID = 3102, Boston
nodeID = 3133, Berlin
nodeID = 3100, Beijing
m = 16, n = 8, b = 2
1 wide-area hop
= reasonable latency
NodeID = 1023, Warsaw
Key = 31033102

Other issues

- Node joining similar to Chord.
- Failure repair automatic.
  - Pastry is self-managed.
- **QUESTION:** What about administrative scalability?

Other issues

- Node joining similar to Chord.
- Failure repair automatic.
    - Pastry is self-managed.
- **QUESTION:** What about administrative scalability?
- Pastry assumes cooperating nodes.