CIS 455/555: Internet and Web Systems

Pastry; Message Queueing

March 13, 2013

Plan for today

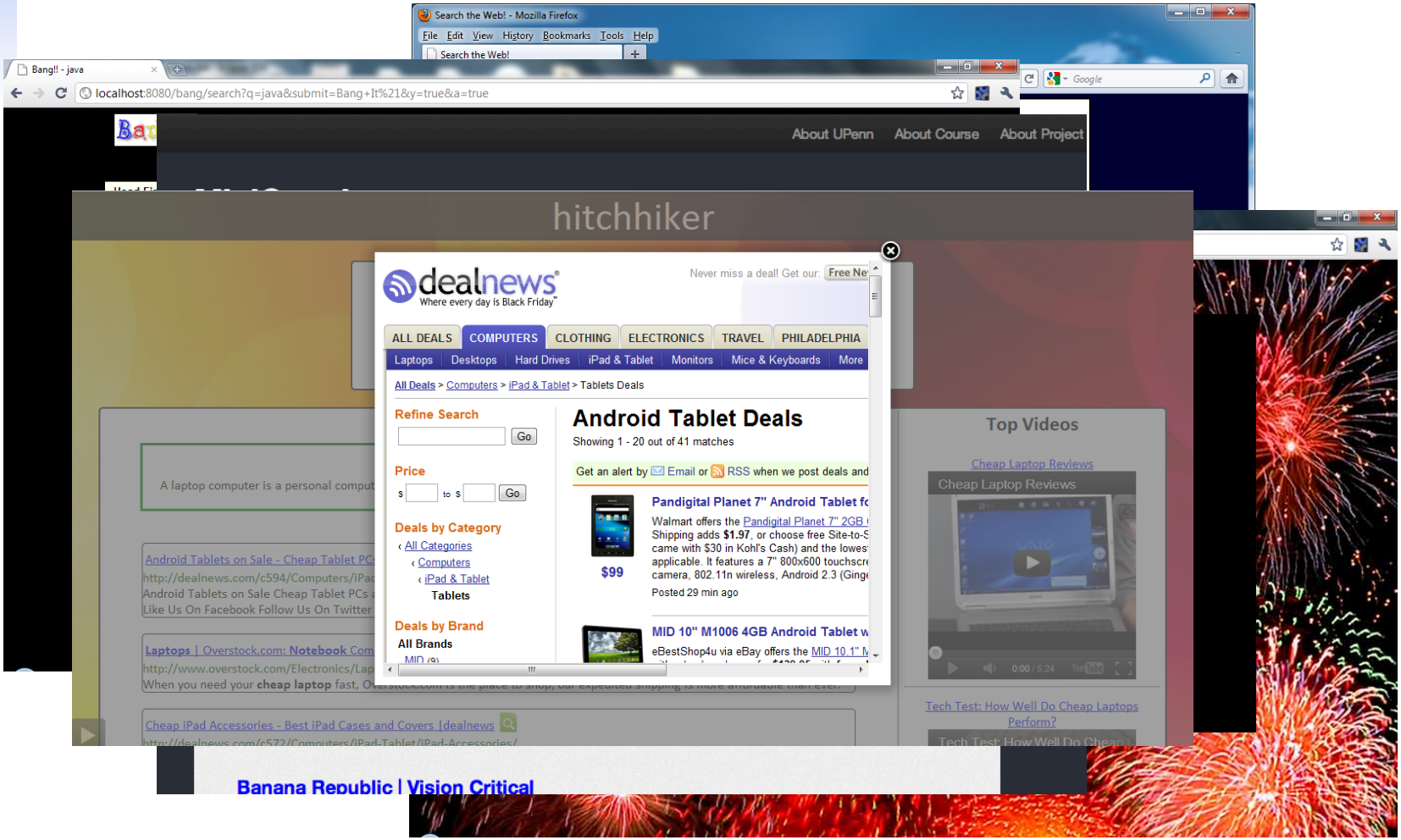
- A few words about the team project
- Pastry
  - Differences to Chord
  - API basics
- Message Queueing
- Remote Procedure Calls
  - Abstraction
  - Mechanism
  - Stub-code generation

NEXT

Relevant for HW3!

The team project

- Task: Build a P2P-based search engine
- Should consist of four components:
  - Crawler, based on your crawler (HW2) and Pastry (HW3)
  - Indexer, based on Pastry (HW3) and BerkeleyDB
  - PageRank, based on MapReduce
  - Search engine and user interface
  - Draft specs are available on the course web page
- Deploy & evaluate on Amazon EC2
  - Need to evaluate performance and write final report
  - Amazon has donated credit codes for this assignment
  - Will send out credit codes soon


Search the Web! - Mozilla Firefox
File Edit View History Bookmarks Tools Help
Search the Web!
Bang!! - java
localhost:8080/bang/search?q=java&submit=Bang+It%21&y=true&a=true
About UPenn
About Course
About Project
hitchhiker
dealnews
Where every day is Black Friday
ALL DEALS
COMPUTERS
CLOTHING
ELECTRONICS
TRAVEL
PHILADELPHIA
Laptops
Desktops
Hard Drives
iPad & Tablet
Monitors
Mice & Keyboards
More
All Deals > Computers > iPad & Tablet > Tablets Deals
Refine Search
Go
Price
Deals by Category
All Categories
Computers
iPad & Tablet
Tablets
Deals by Brand
All Brands
Android Tablet Deals
Showing 1 - 20 out of 41 matches
Get an alert by Email or RSS when we post deals and
$99
Posted 29 min ago
Top Videos
Cheap Laptop Reviews
Tech Test: How Well Do Cheap Laptops Perform?
Banana Republic | Vision Critical

Logistics

- Rough timeline (preliminary):
  - Late March: Begin project planning
  - Early April: Initial project plan due
  - April 29: Official code submission deadline
  - April 30-May 3: Project demos
  - May 7: Final report due (hard deadline!)
- Todo: Form project groups
  - Each team should have 4 members
  - 5-member group requires approval & needs to do some extra credit tasks
  - One person from each group should send me a list of group members by Friday
  - I may have to split or merge some groups

The Google award

- The team with the best search engine will receive an award (sponsored by Google)
  - Criteria: Architecture/design, speed, reliability, quality of search results, user interface, written final report
  - Winning team gets four Android cell phones
  - Winners will be announced on the course web page

Some 'lessons learned' from last year

- The most common mistakes were:
  - **Started too late**; tried to do everything at the last minute
    - You need to leave enough time at the end to a) crawl a sufficiently large corpus, and b) tweak the ranking function to get good results
  - Underestimated amount of **integration work**
    - Suggestion: Define clean interfaces, build dummy components for testing, exchange code early and throughout the project
  - Performance issues
    - Do NOT use Pastry to transfer large amounts of data; use it for small, infrequent coordination messages!
  - Underestimated EC2 deployment
    - Try your code on EC2 as early as possible
  - Unbalanced team
    - You need to pick your teammates wisely, make sure everyone pulls their weight, keep everyone motivated, ...

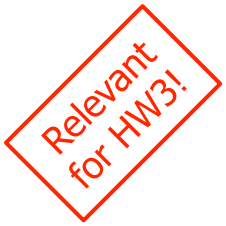
Announcements

- HW2 MS2 is due on March 25th
  - Try to finish by Friday!
    - You can attend office hours / lab sessions
    - You'll have time for testing, and to fix unforeseen problems
  - Why not get started today?
- Second midterm is on April 22nd **at 4:30pm**
  - Likely location: DRL A1
- Reading for today:
  - A. Rowstron, P. Druschel: "Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems"
    - http://www.freepastry.org/PAST/pastry.pdf

Plan for today

- A few words about the team project
- Web services (continued) 

- Pastry
  - Differences to Chord
  - API basics
- Message Queueing



The Facebook API

- What you can do:
  - Read data from profiles and pages
  - Navigate the graph (e.g., via friends lists)
  - Issue queries (for posts, people, pages, ...)
  - Add or modify data (e.g., create new posts)
  - Get real-time updates, issue batch requests, ...

- How you can access it:
  - Graph API
  - FQL
  - Legacy REST API

JSON

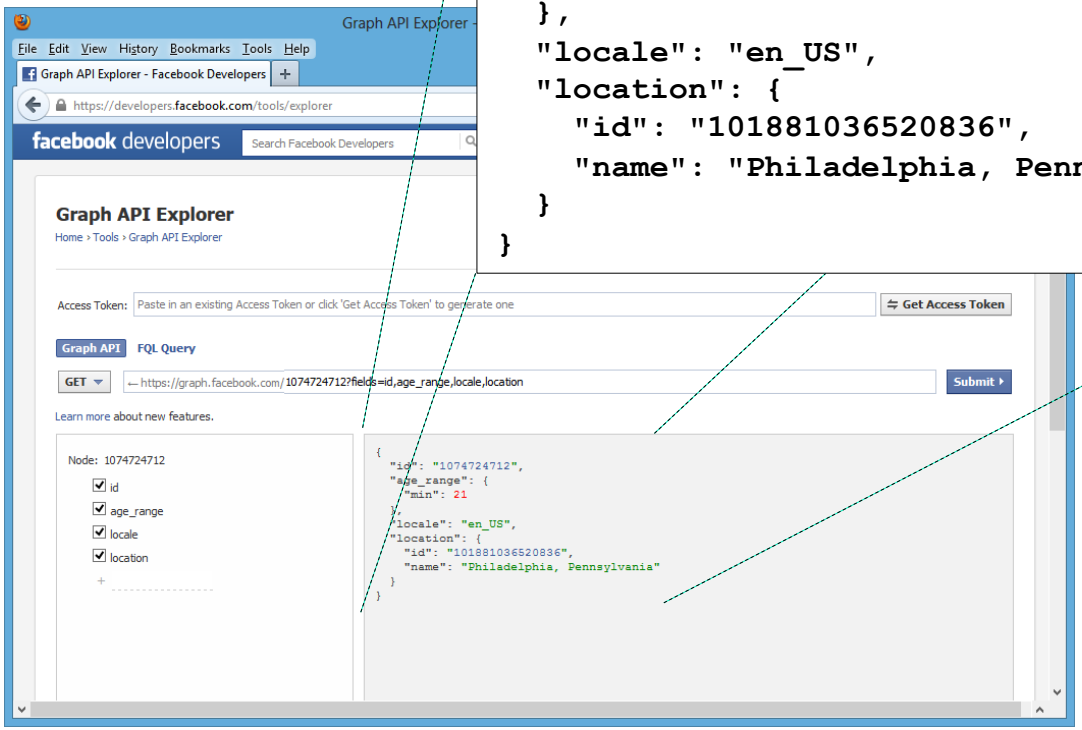
"Object": Unordered collection of key-value pairs

```
{
  "firstName": "John",
  "lastName": "Smith",
  "age": 25,
  "address": {
    "streetAddress": "21 2nd Street",
    "city": "New York",
    "state": "NY",
    "postalCode": 10021
  },
  "phoneNumber": [
    { "type": "home", "number": "212 555-1234" },
    { "type": "fax", "number": "646 555-4567" }
  ]
}
```

Array (ordered sequence of values; can be different types)

- Another standard for data interchange
  - "JavaScript Object Notation"; MIME type application/json
  - Basically legal JavaScript code; can be parsed with eval()
    - Caution: Security!
  - Often used in AJAX-style applications
  - Data types: Numbers, strings, booleans, arrays, "objects"

The Graph API

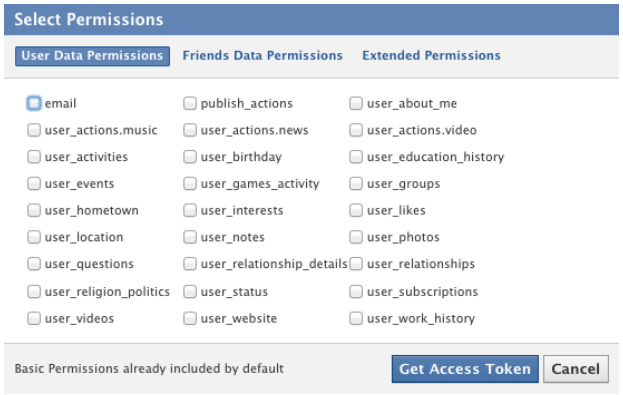



```
{
  "id": "1074724712",
  "age_range": {
    "min": 21
  },
  "locale": "en_US",
  "location": {
    "id": "101881036520836",
    "name": "Philadelphia, Pennsylvania"
  }
}
```


- Requests are mapped directly to HTTP:
  - https://graph.facebook.com/(identifier)?fields=(fieldList)
- Response is in JSON

The Graph API

- Uses several HTTP methods:
  - GET for reading
  - POST for adding or modifying
  - DELETE for removing
- IDs can be numeric or names
  - /1074724712 or /andreas.haeberlen
  - Pages also have IDs



- Authorization is via 'access tokens'
  - Opaque string; encodes specific permissions (access user location, but not interests, etc.)
  - Has an expiration date, so may need to be refreshed

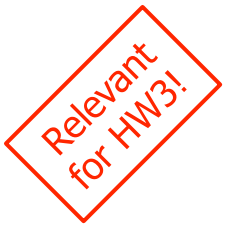
Other API options

- Facebook Query Language (FQL)
  - SQL-style queries over the data provided via the Graph API
  - Example: SELECT name FROM user WHERE uid=me()
  - Supports 'multi-queries' (JSON-encoded dictionary of queries)
  - 72 different tables are available for querying
- Legacy REST API
  - See description of REST in the previous lecture
  - In the process of being deprecated
  - Large number of methods available
    - Examples: friends.get, comments.add, status.set, video.upload, ...

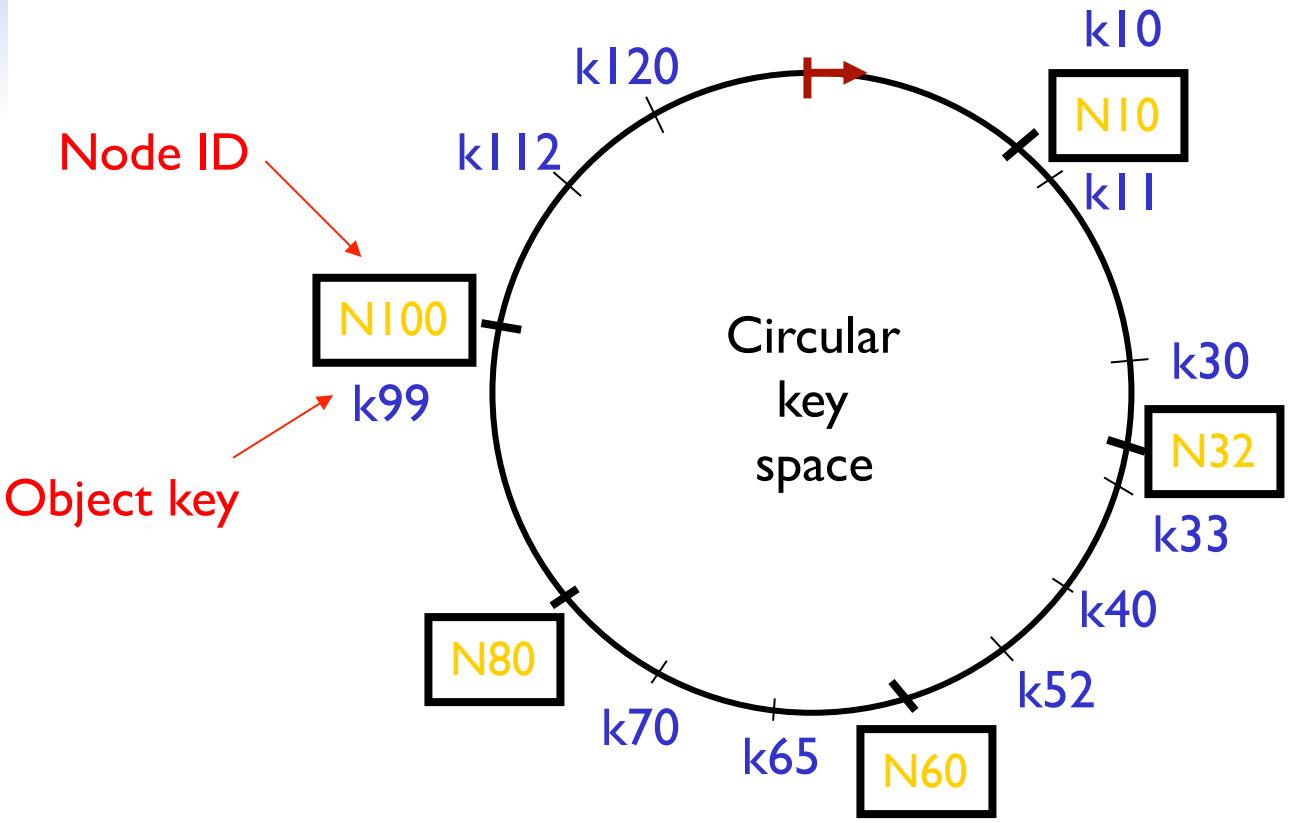
Plan for today

- A few words about the team project
- Pastry 

  - Differences to Chord
  - API basics
- Message Queueing



Remember Chord?



- Large circular key space; objects and nodes have keys
- Key-based routing: route(message, key)

From Chord to Pastry

- What we saw was the basic data algorithms for the Chord system
- Pastry is a slightly different:
  - It uses a slightly different mapping mechanism (closest neighbor, not successor)
  - It has some extra features (leaf set, proximity-aware routing)
  - It allows for replication of data and finds the closest replica
  - It's written in Java, not C
  - … And you'll be using it for HW3 and the final project!

FreePastry

- An open-source Java implementation of Pastry
  - Main web page:
    http://www.freepastry.org/
  - Trac-Wiki:
    https://trac.freepastry.org/
  - Tutorial:
    https://trac.freepastry.org/wiki/FreePastryTutorial
  - Frequently Asked Questions:
    https://trac.freepastry.org/wiki/FreePastryFAQ
  - Discussion group archives:
    https://mailman.rice.edu/pipermail/freepastry-discussion-l/
  - Publications (if you're interested):
    http://www.freepastry.org/pubs.htm

Pastry API Basics

- Keys/Node-IDs implement rice.p2p.commonapi.Id
  - Typical key space is $0..2^{160}-1$ (SHA-1)
  - Generated by factories (NodeIdFactory): CertifiedNodeIdFactory, RandomNodeIdFactory, IPNodeIdFactory - we'll use the latter
  - Each node has a NodeId (which implements Id) + Ids are used for routing
- Nodes are logical entities
  - A single physical machine can have more than one (virtual) node
  - Created by a PastryNodeFactory - in our case, SocketPastryNodeFactory (but there are others, e.g., for local simulations)
- All Pastry nodes have built in functionality to manage routing
  - Derive from "common API" rice.p2p.commonapi.Application

Creating a P2P Network

```
public DistTutorial(int bindport, InetSocketAddress bootaddress, Environment env) {
  NodeIdFactory nidFactory = new RandomNodeIdFactory(env);
  PastryNodeFactory factory = new SocketPastryNodeFactory(nidFactory, bindport, env);
  PastryNode node = factory.newNode();
  /* register your applications here */
  node.boot(bootaddress);
  synchronized(node) {
    while(!node.isReady() && !node.joinFailed()) {
      // delay so we don't busy-wait node.wait(500); abort if can't join
      if (node.joinFailed())
        throw new IOException("Could not join the FreePastry ring. "+
        "Reason:"+node.joinFailedReason());
    }
  }
  System.out.println("Finished creating new node "+node);
}
```

- Example code in DistTutorial.java
  - Creates a new node, which joins the overlay ("ring")
  - Need to provide address of an existing member of the ring ("bootstrap node"); if no ring exists yet, it starts a new one
- No need to call a simulator – this is real!
  - For more information, see FreePastry tutorial: https://trac.freepastry.org/wiki/tut_lesson3

Pastry API basics

- Based on a model of routing messages
  - Derive your message from rice.p2p.commonapi.Message
  - Every message gets an Id corresponding to its key
- Concept of node handles (NodeHandle class)
  - Can be used to talk directly to a specific node (why would you need this?)
  - Internally, has a NodeId, and IP address, and a port number
- Concept of endpoints (Endpoint class)
  - You write an application (rice.p2p.commonapi.Application) and register it with the Pastry node to get one
  - Nodes can have multiple endpoints; provide string as identifier

Routing API in Pastry

- Call endpoint.route(id, msg, hint) to send a message
  - If id is given and hint is null, use key-based routing
  - If hint (a NodeHandle) is given and id is null, send message directly to node
  - If both are given, key-based routing will be used, and hint will be first hop
  - Not reliable -- can lose messages (what does this mean for your app?)
- At each intermediate point, Pastry makes an upcall (forward) to the corresponding application
  - You are given the message that is being routed
  - You can read out (or change) the key, the contents, and the NodeHandle of the next hop
- At the end, Pastry makes a final upcall (deliver) to your application
  - Called on the node whose NodeId is closest to the message's Id
  - Is given the message and the Id

Transferring large amounts of data

- FreePastry uses a single TCP connection for its message-based traffic
  - If you route huge messages, this will stall FreePastry's maintenance traffic (and, obviously, other messages)
  - Likely result: Poor performance
- To transfer large amounts of data, use the application-level socket interface
  - Sender can open a direct connection to the receiver, without intermediate hops (requires a NodeHandle, though)
  - https://trac.freepastry.org/wiki/tut_app_sockets
- Use messages only for coordination
  - Don't send too many; "congestion control" may be needed

Making IDs

- Pastry has mechanisms for creating node IDs itself
- Obviously, we need to be able to create IDs for keys
- Need to use java.security.MessageDigest:

```
MessageDigest md = MessageDigest.getInstance("SHA");
byte[] content = myString.getBytes();
md.update(content);
byte shaDigest[] = md.digest();

rice.pastry.Id keyId = new rice.pastry.Id(shaDigest);
```

Creating a DHT abstraction with Pastry

We want the following:

- put (key, value)
- remove (key)
- valueSet = get (key)

- How can we use Pastry to do this?

Recap: Pastry

- A substrate for decentralized systems
  - Implements key-based routing
  - Similar to Chord, but has some additional features, e.g., leaf sets and proximity neighbor selection
  - Open-source Java implementation available (FreePastry)

- Main FreePastry abstractions:
  - Id and NodeId (keys from a large key space)
  - Endpoint (for sending/receiving messages)
  - Node (has a NodeId, provides endpoints, handles routing)
  - Application (superclass for your own KBR application)

Plan for today

- A few words about the team project ✓
- Pastry ✓
  - Differences to Chord ✓
  - API basics ✓
- Message Queueing ← NEXT

Relevant for HW3!

Why message queueing?

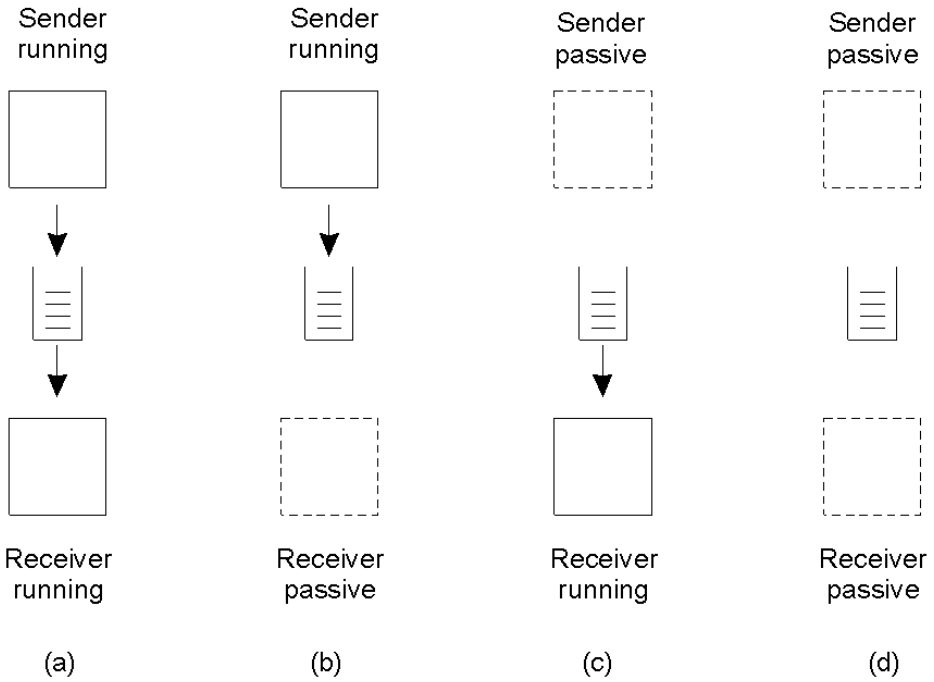
- So far, we have seen synchronous/transient communication
  - Example: REST call to a web service
- But what if applications are more loosely coupled?
  - Maybe the service we're interacting with isn't always available (batch process etc.)

Types of communication

- There are several types of communcation:
  - Persistent vs transient
    - If the sender and the receiver are not running, is the message lost?
  - Synchronous vs asynchronous
    - Does the sender continue immediately, without knowing that the message has been accepted?

- What would be an example of...
  - Synchronous, transient communication?
  - Synchronous, persistent communication?
  - Asynchronous, transient communication?
  - Asynchronous, persistent communication? ← Need queues! Hence MQM
  - Which of these are tightly/loosely coupled?

Message-Queuing Model



- Apps communicate by inserting messages into queues
  - Sender only knows that message has been inserted into queue, not that it has been seen or processed by the recipient (real-world analogies?)
  - Four combinations; see above
- Implemented e.g., by IBM WebSphere MQ, Oracle AQ, ...
  - Similar: Amazon SQS

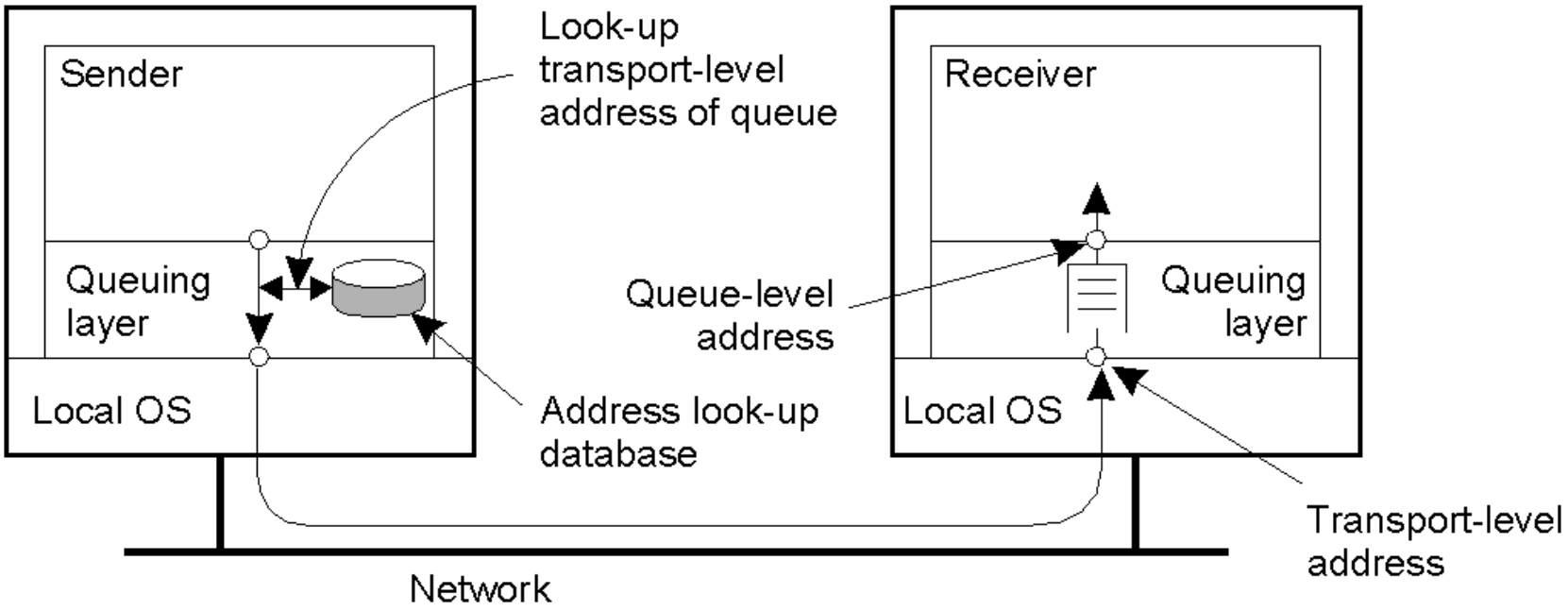
Message-Queuing API

- Basic interface to a queue in a message-queuing system.

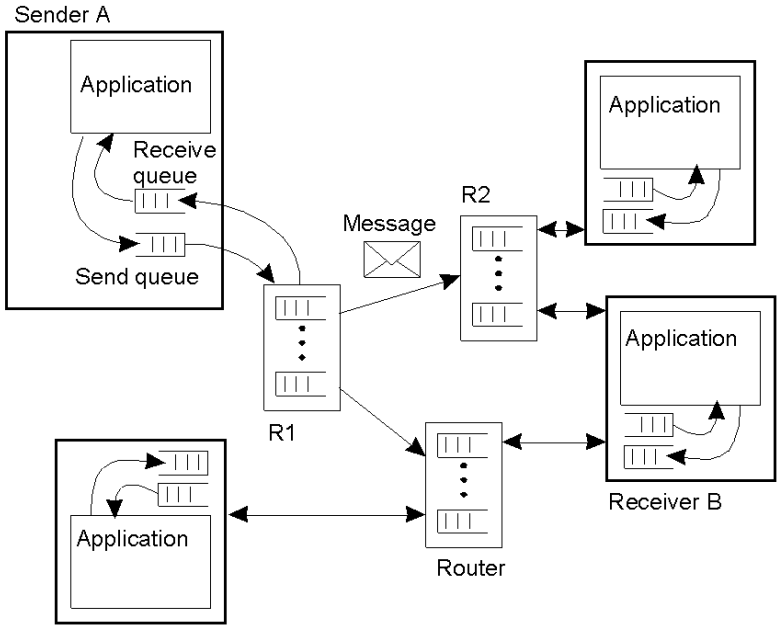
Primitive	Meaning
Put	Append a message to a specified queue
Get	Block until the specified queue is nonempty, and remove the first message
Poll	Check a specified queue for messages, and remove the first. Never block.
Notify	Install a handler to be called when a message is put into the specified queue.

General Architecture of a MQ System

- The relationship between queue-level addressing and network-level addressing.



MQ System with routers



- Problem: Each queue manager has to be able to find the location of any given queue
  - Complexity / scalability challenges
- Solution #1: Directory service
- Solution #2: Relays / routers

Benefits of message queueing

- Allows both synchronous (blocking) and asynchronous (polling or event-driven) communication
- Ensures messages are delivered (or at least readable) in the order received
- The basis of many transactional systems
  - e.g., Microsoft Message Queue (MMQ), IBM MQseries, etc.