CSE 515 — Winter 2004

Introduction

Class 1

Class Format and Content

You are here to

1. Learn about distributed systems (information)
2. Learn how new results are discovered in distributed systems (meta-information)
   – *e.g.*, impossibility results
3. Learn how to find out more about a new system
   – *e.g.*, what questions to ask

Class format

Reading

- There will be a lot of it, and it's not optional.
- There will be a fair amount of writing too!

Before Class

- Students and instructor read textbook chapters or papers *before* class
  - write chapter summary or questions as homework

In Class:

- Students and instructor summarize content of the reading
- *Everyone participates* in discussion based on provocative questions supplied by the moderator
  – the moderator is not necessarily an expert
  – the point of the discussion is to explore possibilities, not to show what you know
    - open-ended questions are better than y/n questions
  – 20% of your grade will depend on this interaction.

Homework

1. Introduce yourself
2. Performance of RPC or RMS middleware
3. Implementation of RMS or similar
4. Design of distributed application (in groups)
5. Implementation of distributed Application (in groups).
   – presentation and demo

Class Web Page

http://www.cse.ogi.edu/cse515

- Use it!
- Grading policy, readings, course schedule
- Will be updated as the quarter progresses

Questions

What is a Distributed System?

An *integrated* computing or information facility, that is

- built out of many computers
- that operate concurrently
- that are physically distributed (= have their own failure modes)
- and have independent clocks
- but are linked by a network.

Why do we have them?

- Problem is decentralized
  - teleconferencing
  - coordinating automated shop-floor.
- Distributed architecture adds value to solution
  - *e.g.*, bank with many branches
- High value placed on adaptability
  - a distributed system can usually be evolved piecemeal to meet new requirements.

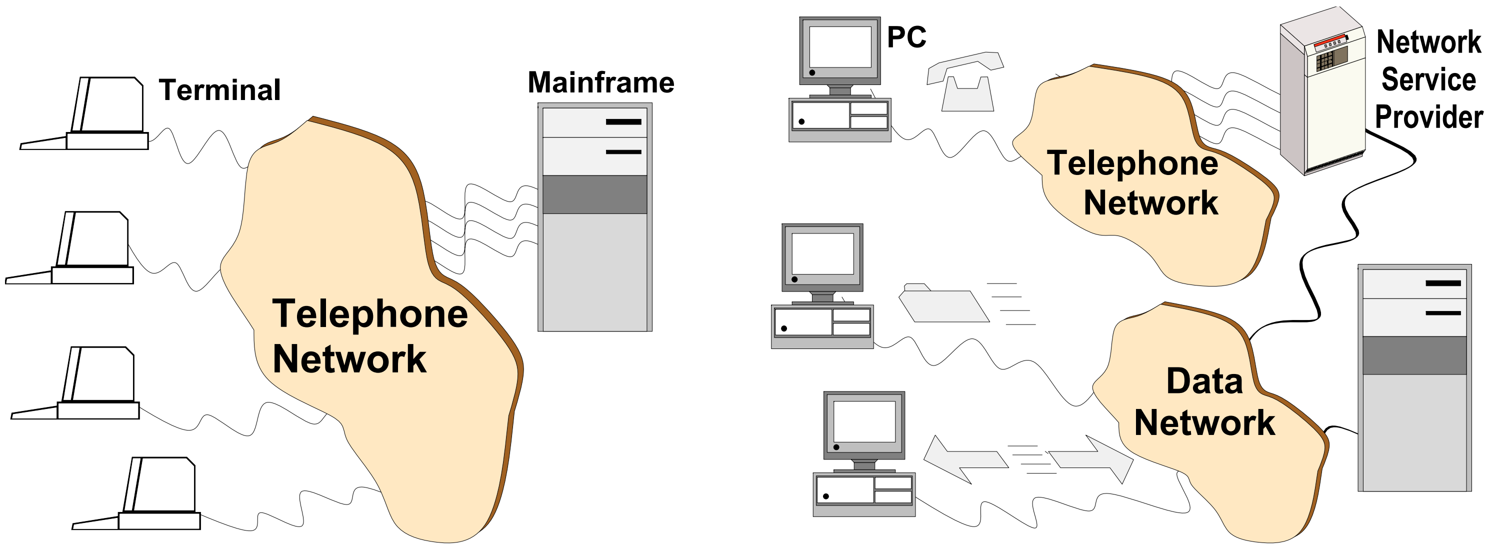
What Values can distribution Add?

- Reliability
  - What's the relationship between the reliability of a system and its size?
- Expandability
  - a well-architected system should allow for modular growth
- Cost effectiveness
  - many small computers can be cheaper than one large one
  - beware the cost of management!

A Short History of Distributed System

1. Remote Access
2. File Sharing and Memory Sharing
3. Remote Access Revisited
4. Client-server
5. Client-server-backend (3-tier)
6. Mobile code
7. Mobile site
8. Event-based

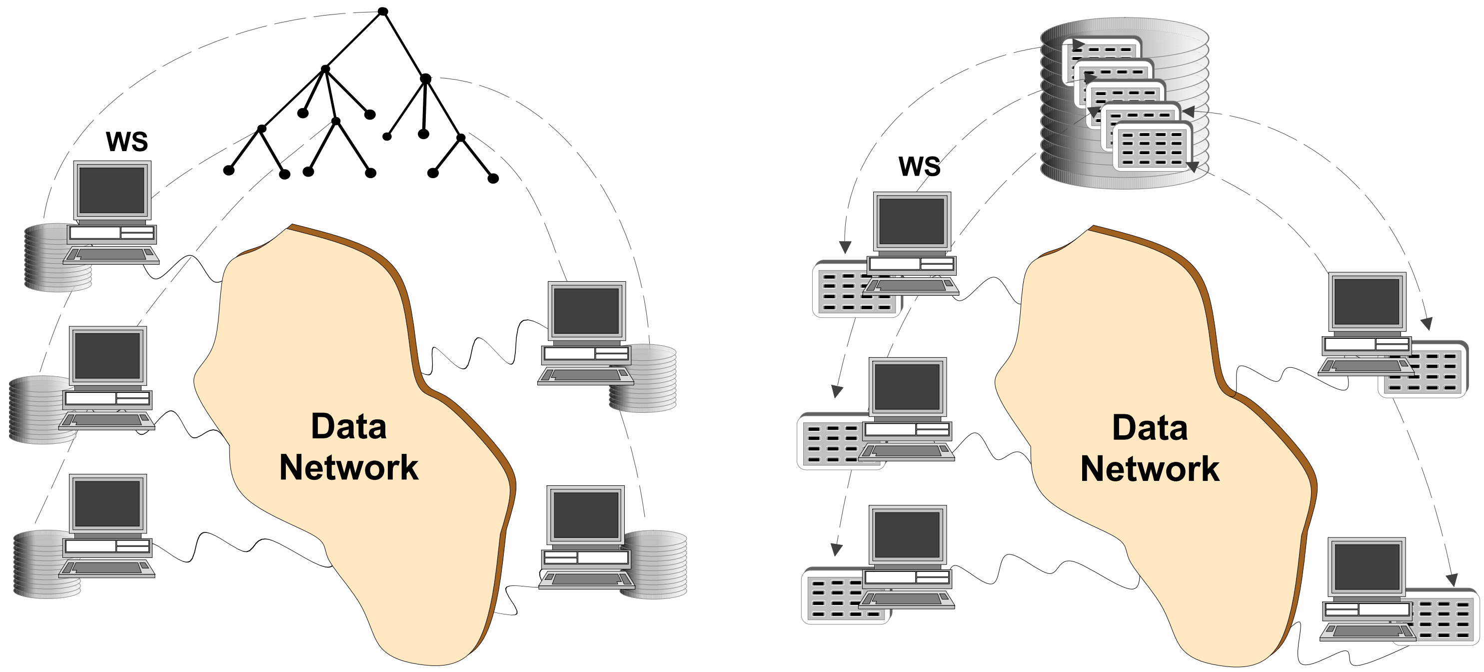
Remote Access



(a) *(b)*

- Not really a distributed system at all
  - All computing takes place on the Mainframe

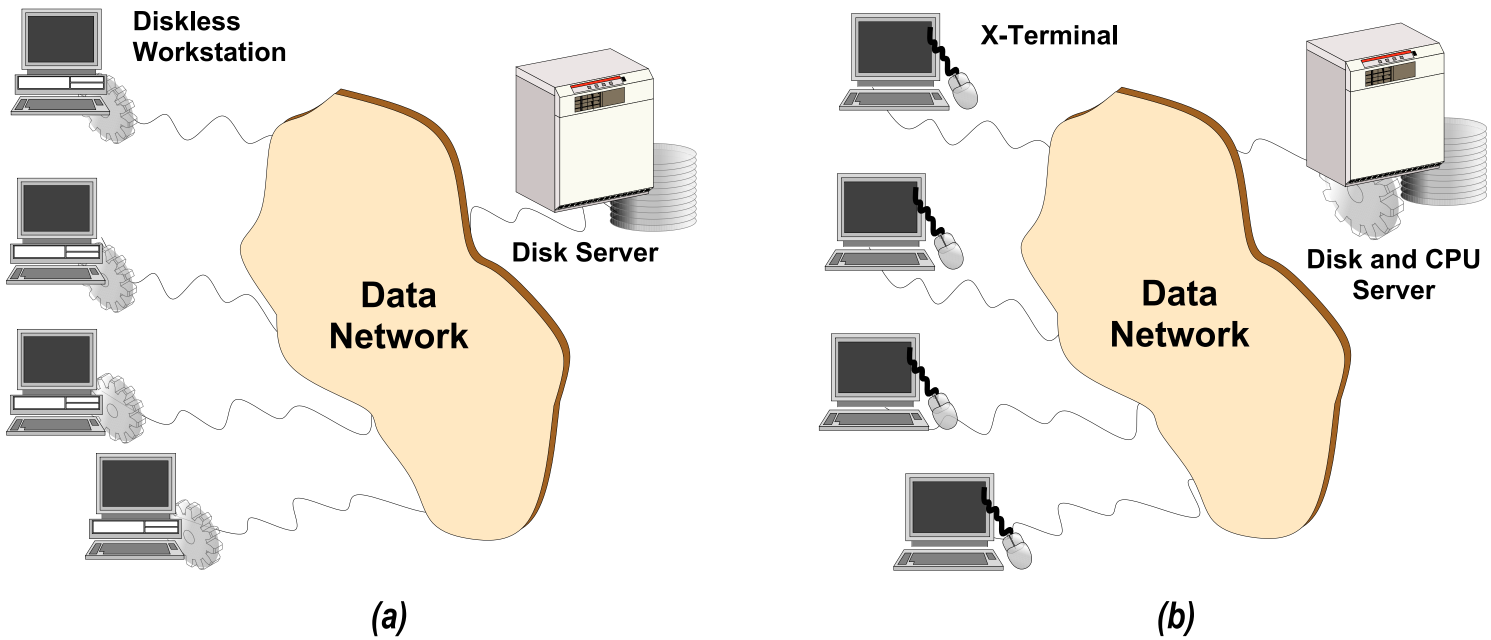
File Sharing and Memory Sharing



(a) *(b)*

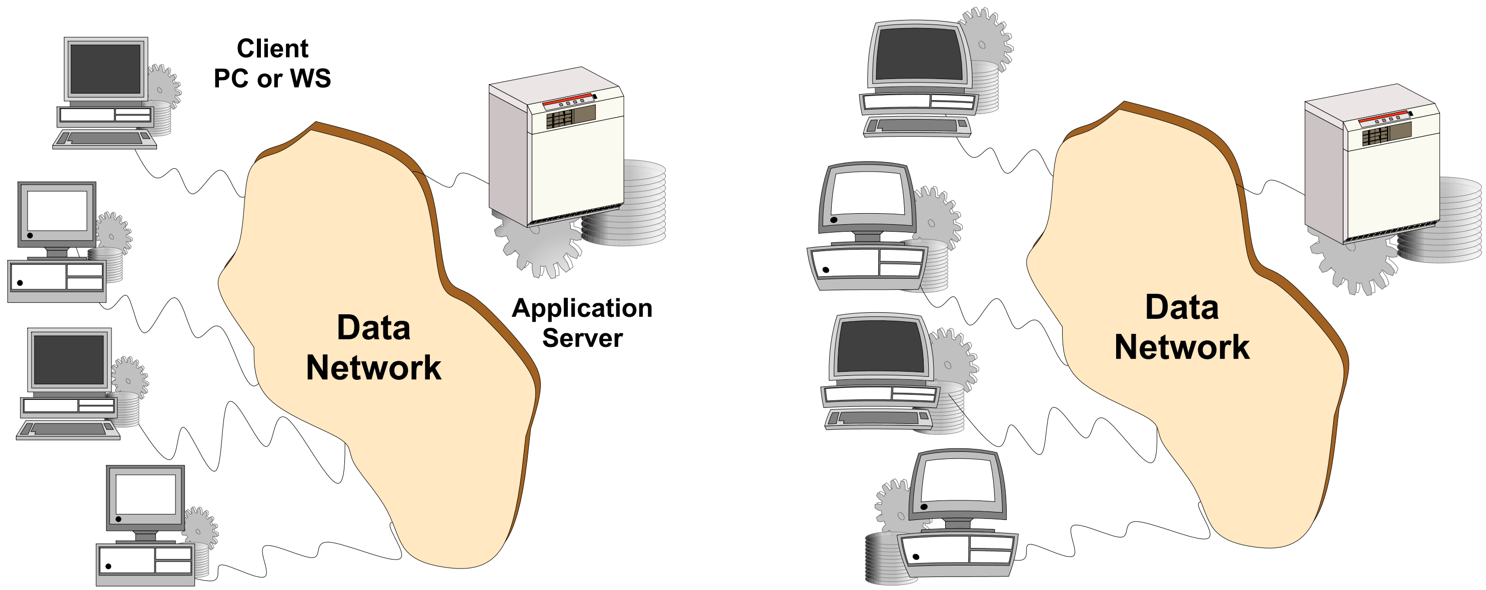
- WS contribute a part of a shared, globally accessible, file system or memory

Remote Access Revisited



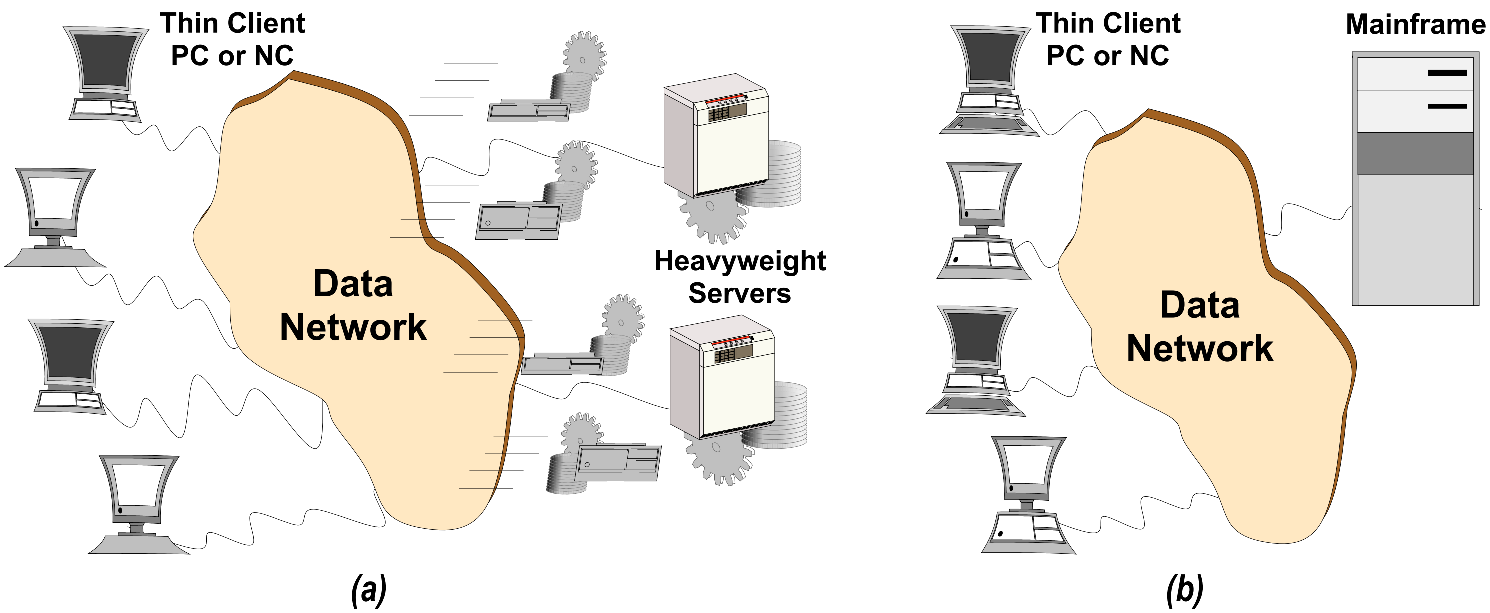
- Disks are expensive! Hence:
  - diskless workstations: cheap and manageable
  - X-terminals: like teletypes on a time-shared system

Client-server

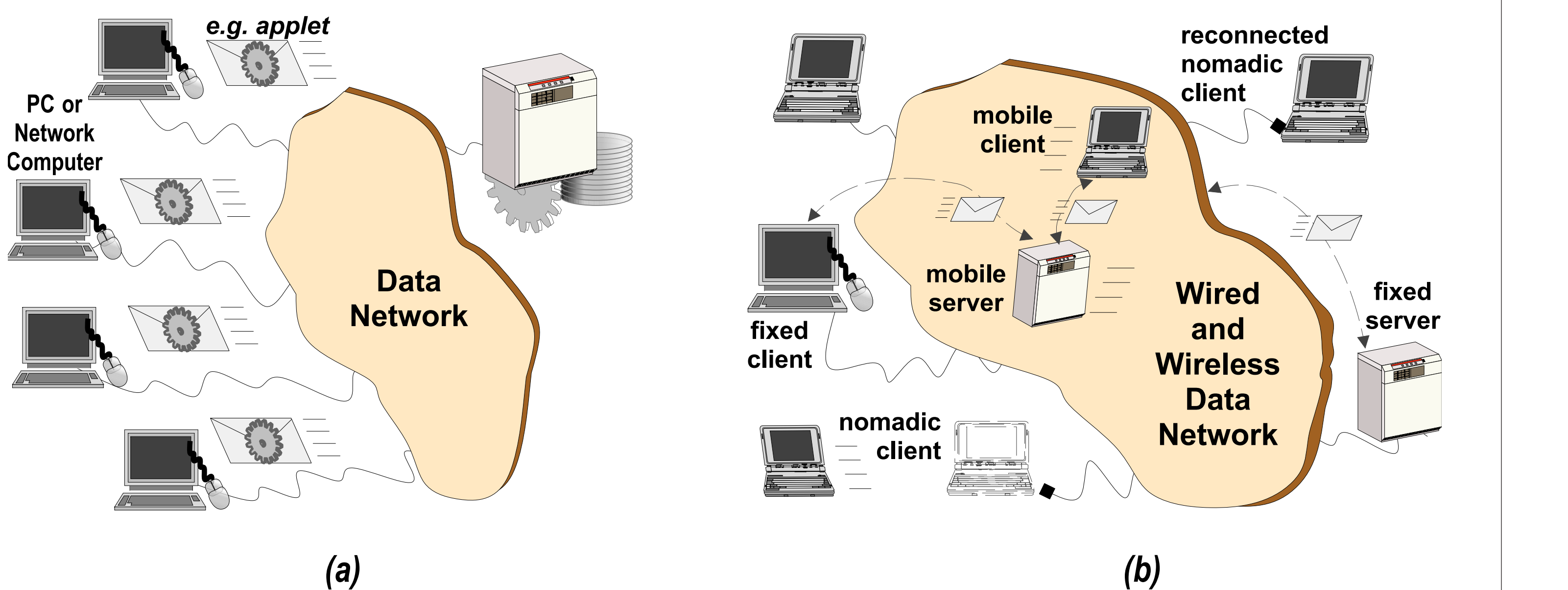


(a) (b)

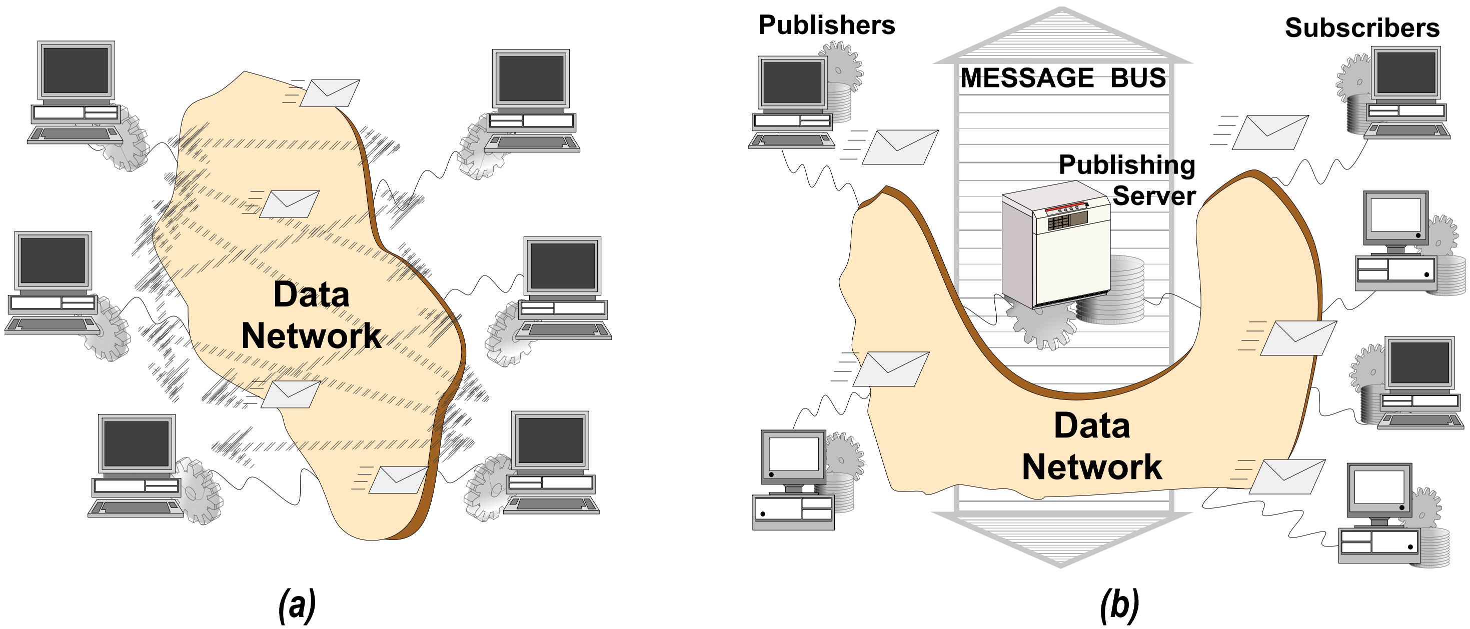
Client-server-backend (3-tier)



Mobile code & Mobile sites



Event-based



- Multicast, *i.e.*, group communication
  a. Peer-to-peer (a.k.a. multipeer) interactions
  b. Central server implements message bus

Commonalities

- What do all of these architectures have in common?

- How do they differ?

Formal Notions and Notations

(See Veríssimo & Rodrigues §1.4)

1. Events and timing
2. Timing diagrams
3. Global state
4. Safety, Liveness & timeliness

Events & Timing

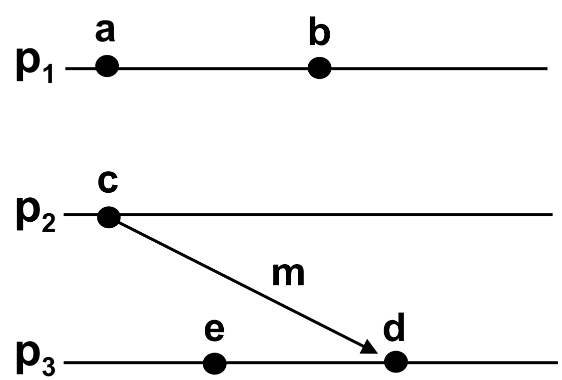
Models of Distributed Systems

- Often convenient to abstract away from physical machines and processes
  - Consider a system of N *participants* or *processes*
- Participant p executes events $e_p^1, e_p^2, \ldots, e_p^i$
  - event e_p may modify the state of p
- $t(e)$: "absolute" time at which e occurred

Timing diagrams

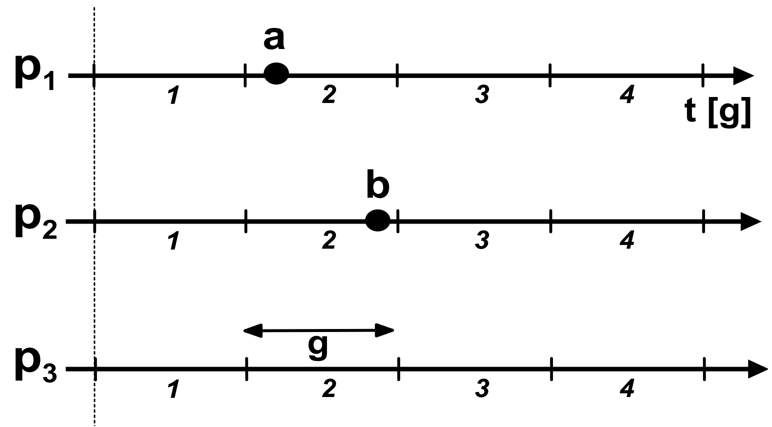
Space-time Diagram

- Three participants p_1, p_2 and p_3
  - p_1 executes local events a and b
  - p_2 executes c, the send of message m to p_3
- Does event c precede e?
  - What about a and b, c and d?



Lattice Diagram

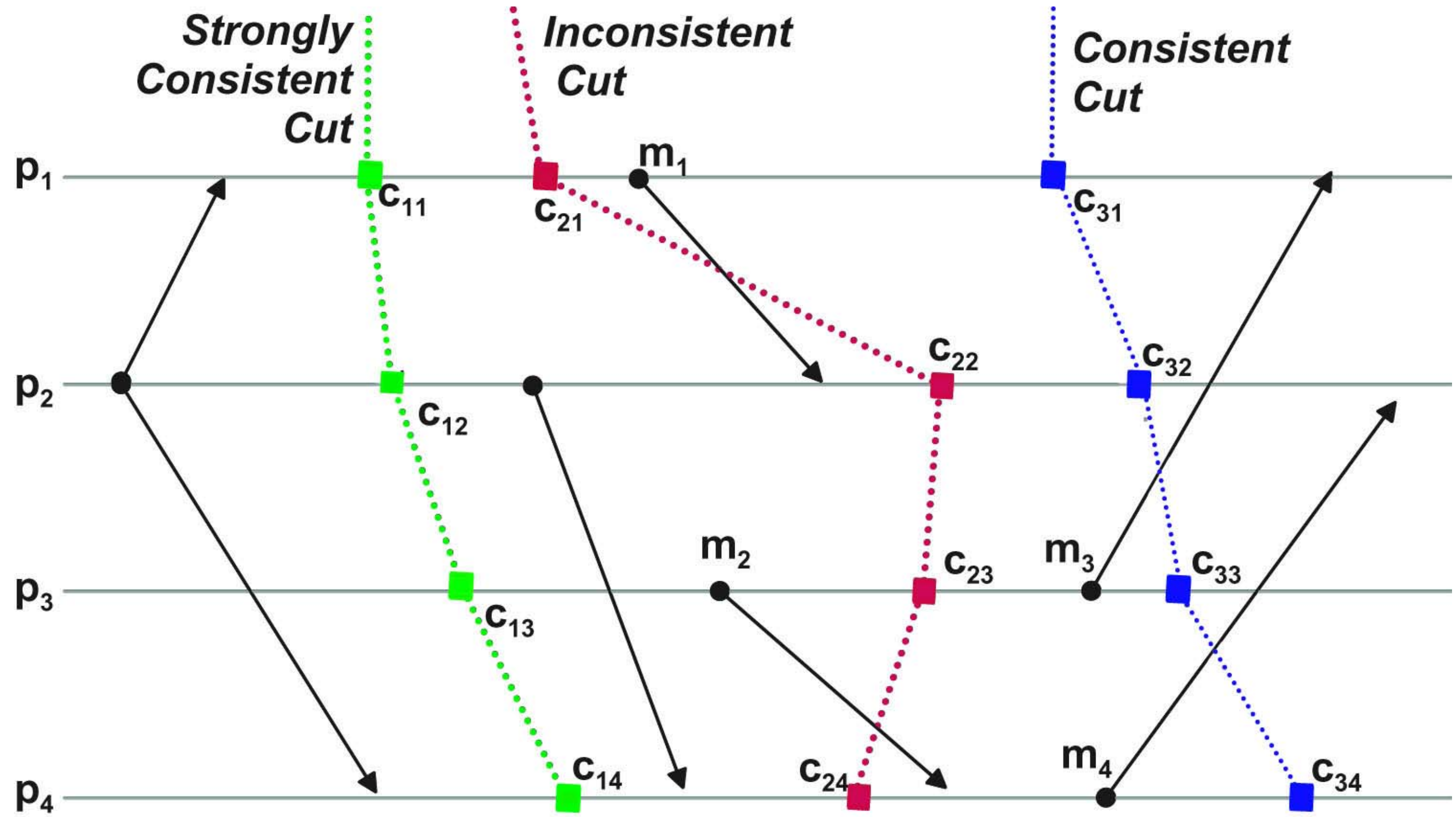
- Even if we can timestamp events, we still may not know if event a precedes event b
  - timestamps have a granularity g that may exceed the duration $a - b$



Global state

- *Conceptually*, the global state is simple:
  - If at time t each participant p_i has local state S_i, then the global state S is given by $S = [S_1\ S_2\ S_3\ \ldots]$
  - S_i must include all messages sent by p_i
- But: how can we know the state of each participant at time t?
  - Only by sending messages, which will change the state …

- Consistent Cut protocols do just that



  – What's wrong with the inconsistent cut?

Safety, Liveness & Timeliness

- Safety properties say that some bad thing does not happen, *e.g.*,
  - Deadlock does not occur
  - If message m is delivered to process p_i, then it will also be delivered to $p_1, p_2, \ldots, p_{i-1}, p_{i+1}, \ldots, p_n$
- Liveness properties say that some good thing will happen eventually, *e.g.*,
  - If message m is retransmitted enough times, it will eventually be delivered to process p_i

- Timeliness properties add real-time constraints to liveness, *e.g.*,
  - Message m will be delivered within 100 ms of the time that it is transmitted.

Naming

- Names are used by people
  - They can refer to hardware (*e.g.*, printers), services (*e.g.*, currency conversion), people (*e.g.*, e-mail addresses) and almost anything else.
  - Names may be
    - globally unique, or
    - context-dependent, *i.e.*, unique only within some context
      - *e.g.*, *black* is unique within the context *cse.ogi.edu*
      - the context must be supplied or inferred to give meaning to the name

- A context-dependent name can be turned into a global name by qualifying it with its context
- Names may be
  - pure (devoid of inherent meaning), *e.g.*, *Pat*, or
  - impure (some information about the object can be extracted from the name), e.g., *ISBN 0-7923-7266-2*, *http://www.cse.ogi.edu/class/cse515*

- Addresses are used by computers
  - IP addresses are an example, *e.g.*, *129.95.40.02:20*
  - Addresses at one level of abstraction may be regarded as names at a lower level
    - is *black@cse.ogi.edu* a *name* or an *address*?

Name to Address Translation

- We use names in preference to addresses:
  - Names are easier to remember
  - Names can be independent of the protocol used to access the object, or the object's location, e.g. *Andrew* is better than *503 690 1250*
  - Names can refer to a group with changing composition, e.g., *cse515@cse.ogi.edu*
  - The meaning of a name can change over time, e.g., *library printer*
- What do we need to do with a Name?

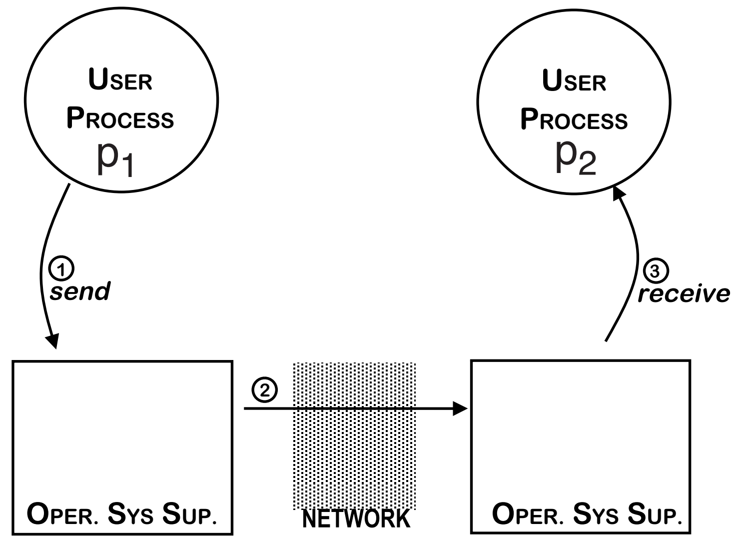
Name Service

- A *Name Service* is able to a *resolve* a name into an address
  - *ns.lookup(name)* $\rightarrow$ *address*
  - *ns.bind(name, address)*
  - *ns.unbind(name)*
- The name service can be implemented in many different ways:
  - centralized server
  - partitioned table
  - fully replicated table
  - broadcast

- For the present, we will ignore the implementation details
  - We will assume that a process can send a *lookup* message to a name *server*, and be told the address
  - How does the process get the address of the name server?
    -
  - Case study of Name Servers in class 13.
    - Issues are availability, scalability and security

Message Passing

- Message passing is the most basic form of interaction in distributed systems
  - process p_1 sends a message to process p_2
  - process p_2 receives the message
- p_1 must have the address of p_2

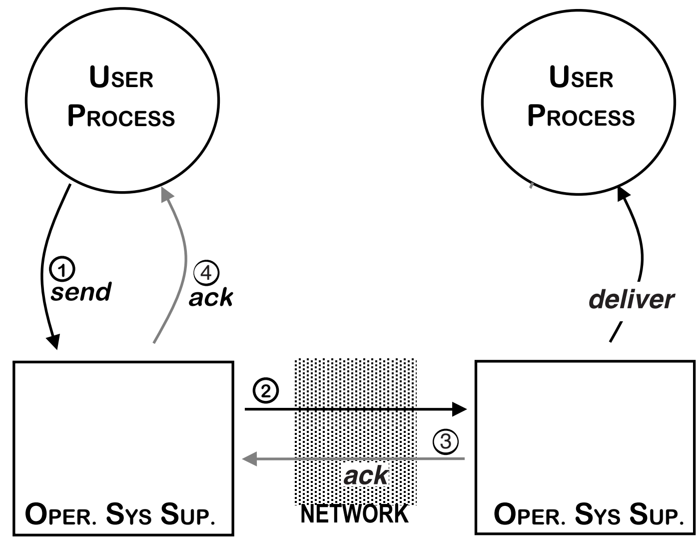


Networking 101

- p_1 and p_2 must agree on a protocol:
  - number, meaning and format of fields in the message
  - byte order, character coding, bits per integer, and such low-level details can't be forgotten
    - version numbers are more important than you think!
- Message send is unreliable
  - Network can drop messages
  - Receiving process can crash
  - Network is asynchronous:
    - time for message is unbounded.

Acknowledging Messages

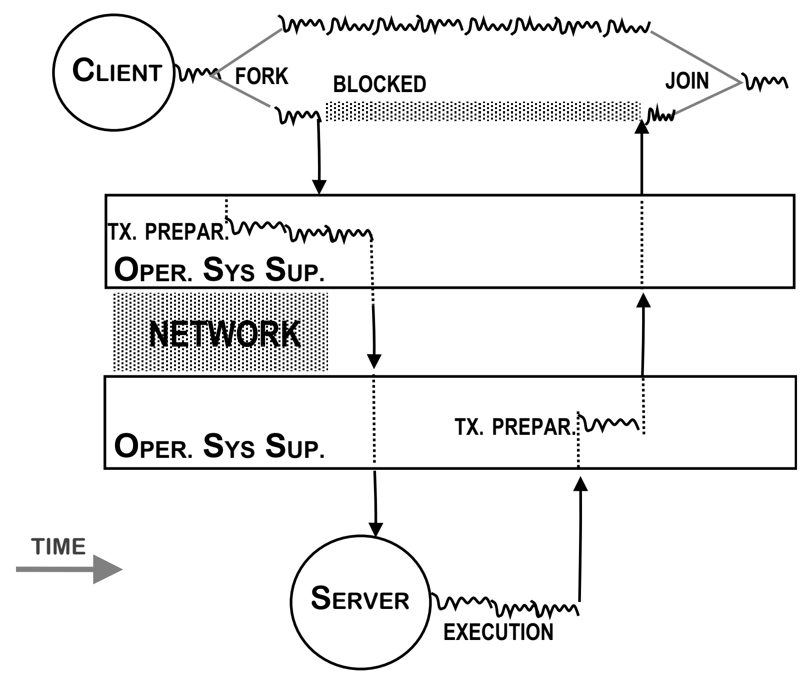
- Should the OS network support automatically acknowledge each message?
  - Not as simple a question as it might seem!



  - What can the sending user process do with the information given by the *ack*?

Blocking *vs.* non-blocking Send

- Should the *send* primitive block until there is a reply?
  - Can the sender do some useful work while waiting?
  - Can the *send* primitive really be made non-blocking?
  - Is the sender multi-threaded?



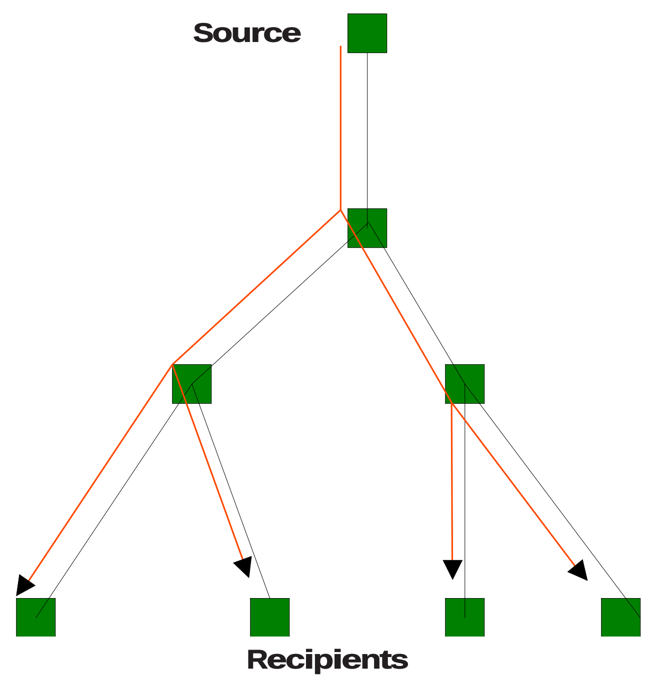
Datagrams *vs.* Reliable Streams

- A Datagram is an unreliable message
  - datagrams may arrive out of order or not at all
  - UDP is a user-level (user-process to user-process) datagram protocol built on top of IP
- A Stream is reliable:
  - either every byte send is delivered, and in the same order, or
  - the stream breaks, telling the user-level sending process that something went wrong
  - TCP is a user-level streaming protocol built on IP

Group Communication

Some distributed system architectures are based on *multicast*

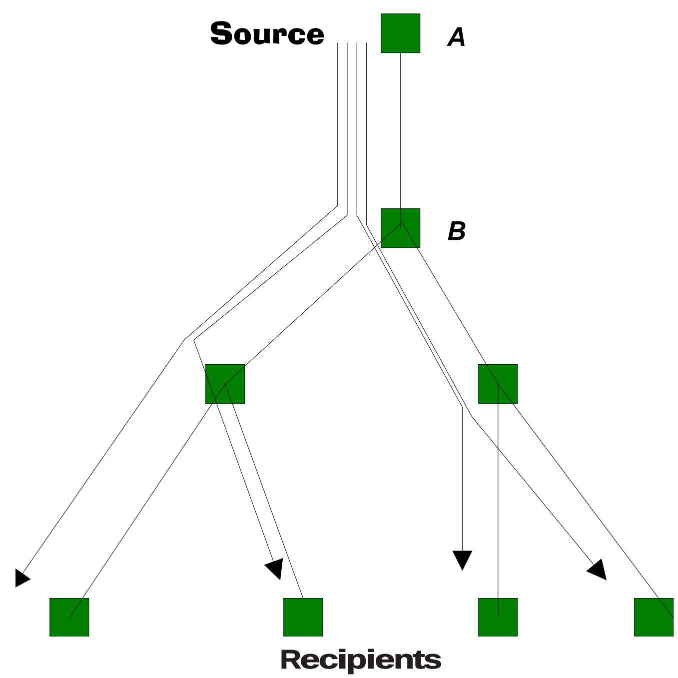
- multicast = sending a single message to multiple recipients
  - different from sending multiple *unicasts*
  - broadcast media and multicast trees



Group Communication

Some distributed system architectures are based on *multicast*

- multicast = sending a single message to multiple recipients
  - different from sending multiple *unicasts*
  - broadcast media and multicast trees



Groups

- Groups can be used as part of the requirements (visible)
  - *e.g.*, send this video to the *viewers* group
- or, as part of the implementation (invisible)
  - *e.g.*, update this *calendar*
  - *calendar* is replicated at the members of a group

Multicast Protocols

Many different multicast protocols are possible:

- Are messages:
  - totally ordered, causally ordered, ordered per sender, or unordered?
- Is delivery reliable?
  - Do all members see the same set of messages?

Multicast Protocols (cont.)

- How are changes in group membership reflected
  - How do group change messages interact with the content messages?
- Are groups *open* or *closed*?
  - Are non-members allowed to send to the group?
- Is flow control provided?

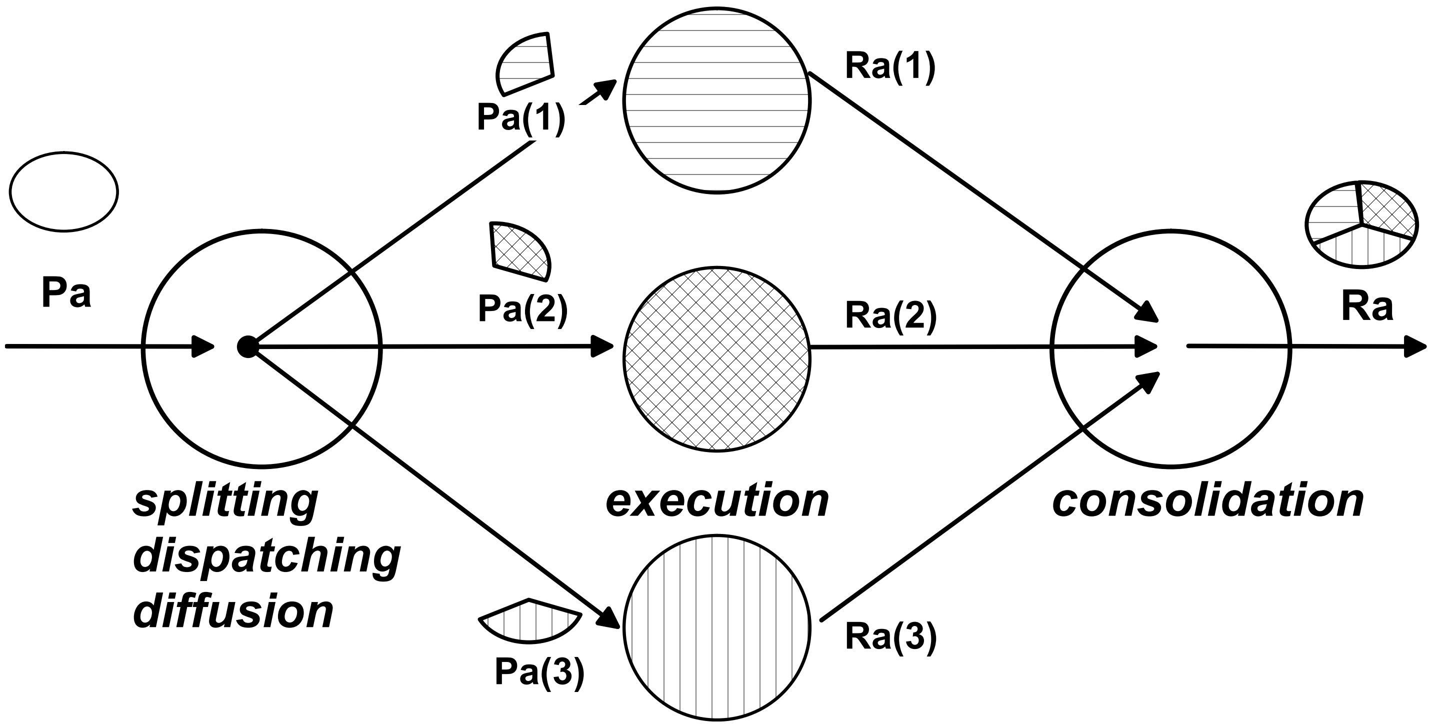
Classes of Activity

There are three classes of distributed activity:

1. Coordination
2. Sharing
3. Replication

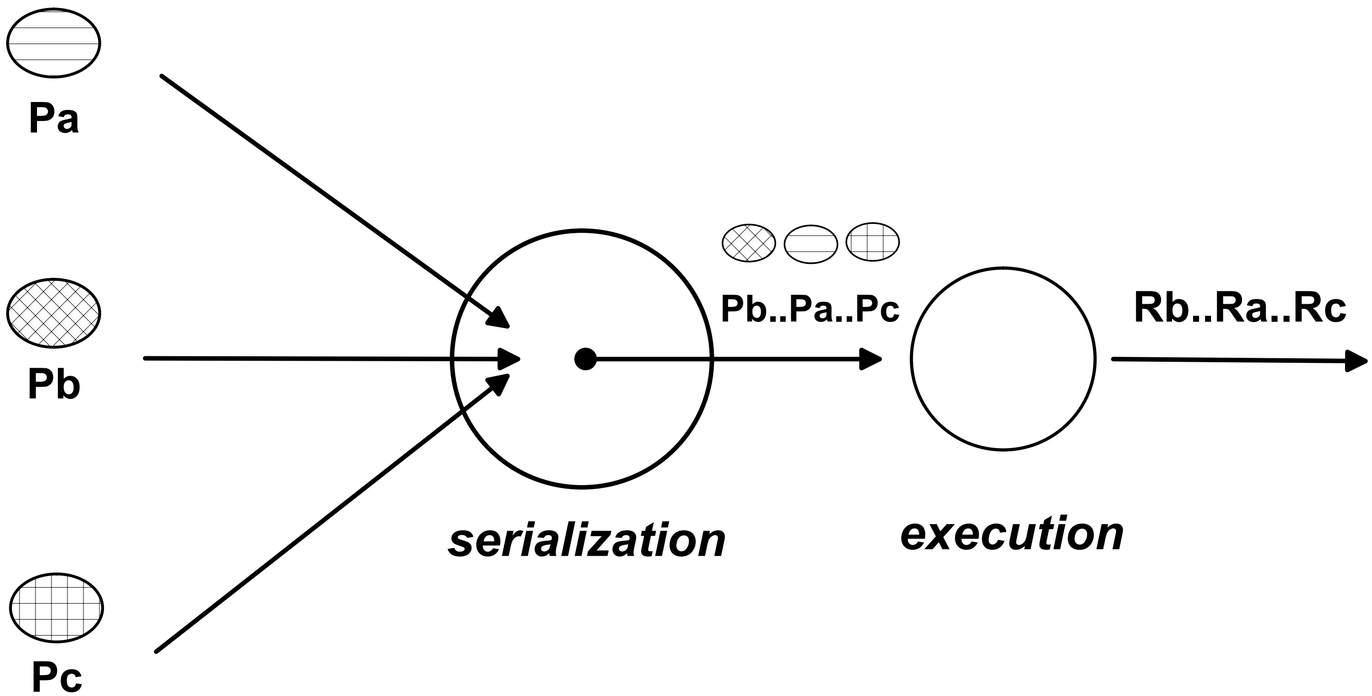
Activity in a real system is usually some combination of these three!

Coordination



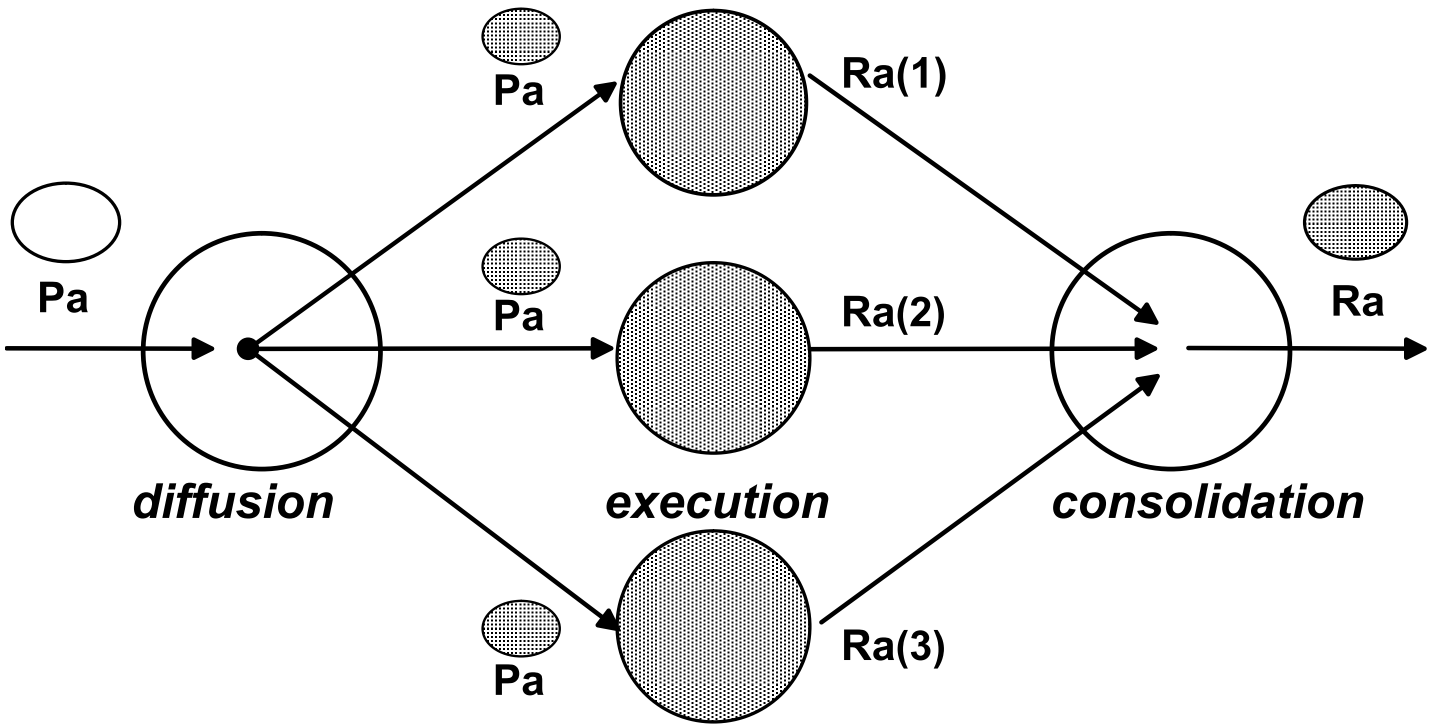
- *e.g.*, booking parts of a trip for a travel agency

Sharing



- *e.g.*, shared web page or printer

Replication



- *e.g.*, updating a replicated file
  - active *vs.* passive replication

Combined Activity