An introduction to distributed systems

Runtimes for concurrency and distribution

Tullio Vardanega, tullio.vardanega@unipd.it

Academic year 2020/2021

Distribution requires transparency

- A distributed system is a set of **independent** computing nodes capable of appearing as a **single** coherent execution platform to applications running on it
  - This requires all coordination communications among those computing nodes to be **transparent** to the application
- **Transparency** is given *when you get to see the intended effects without being exposed to the mechanics that produce them*
  - There exist several dimensions of transparency

Transparency requires openness

ISO/IEC 10746-1:1998, *Open Distributed Processing – Reference model: Overview*

Transparency of	To hide what
Access	Differences in data encoding or in the way to operate on resource data
Location	Where computing resources actually reside (e.g., physical vs logical naming)
Migration / Relocation	Resources may move without the user needing to know in between uses, or even during use
Replication / Transaction	That a resource may exist in multiple coherent copies, or may result from the aggregation of multiple parts
Malfunction	Individual computing nodes may locally fail without this affecting the availability of the resource
Persistency	How writing succeeds regardless of the distance between writer and resource

What is openness

- It is a key prerequisite to **portability** and **interoperability**
- It prescribes all call interfaces to conform to **public** and **stable** specifications
- Such specs have to be
  - **Complete**, so that no details are hidden that may preclude third-party implementations of them
  - **Neutral**, so that they do not impose a single way of implementation
- Interface definition languages (IDL) help achieve such properties

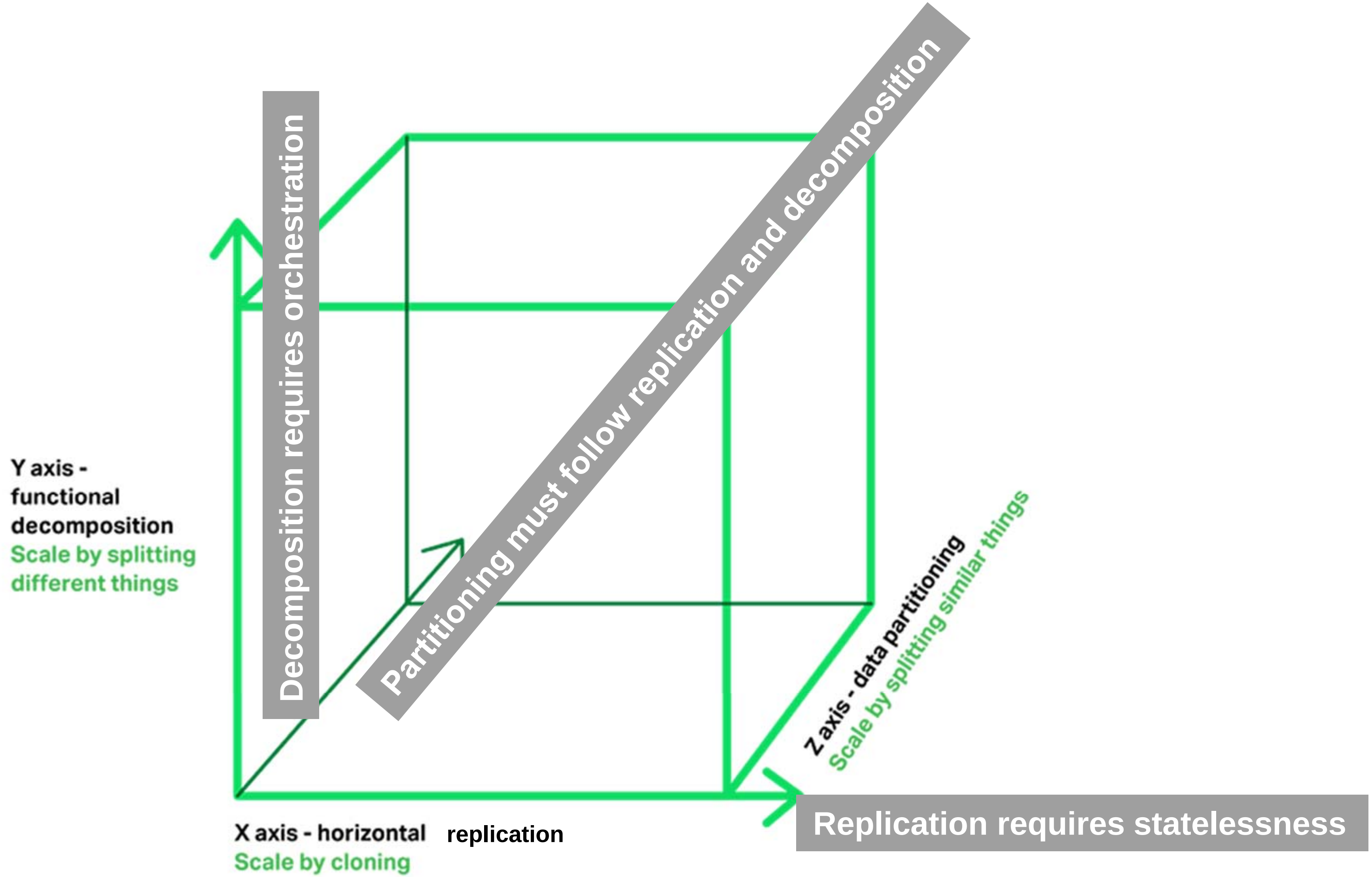
Distribution requires scalability

- Any service provided to distributed clients needs to scale to demand
- Scalability is more easily understood by its negation
  - A system is **not** scalable when it is unable to accommodate increased workload
- A useful definition stipulates it as
  - The ability to handle increased workload *by* ***repeatedly*** *applying a* ***cost-effective*** *strategy for extending system capacity*
    - Without intolerable latency or excessive waste

What is scalability

- Fitness for need with respect to
  - Availability of resources
    - They should never be scarce
  - Physical distance
    - The user should have perception of locality
  - Independence of global view from local issues
    - Issues in handling local, concrete implementation should not determine how a resource is presented to the user
- Where unused resources cost dearly, you want scalability to be **elastic**
  - Not only expanding but also contracting, with equal cost-effectiveness

The scale cube



https://www.nginx.com/blog/introduction-to-microservices/

The opposite of distribution

- Centralization of service
  - All users must refer to a single entry point
    - As in the `HOSTS.TXT` file that mapped hostnames to IP addresses in the ARPANET
- Centralization of resources
  - All the data relevant to a service are kept in a single copy at a single place
    - The opposite of how the DNS (ca. 1985) and Blockchain (ca. 2008) work
- Centralization of algorithm
  - Requiring to know the system state
    - Impossibly burdensome to compute and maintain

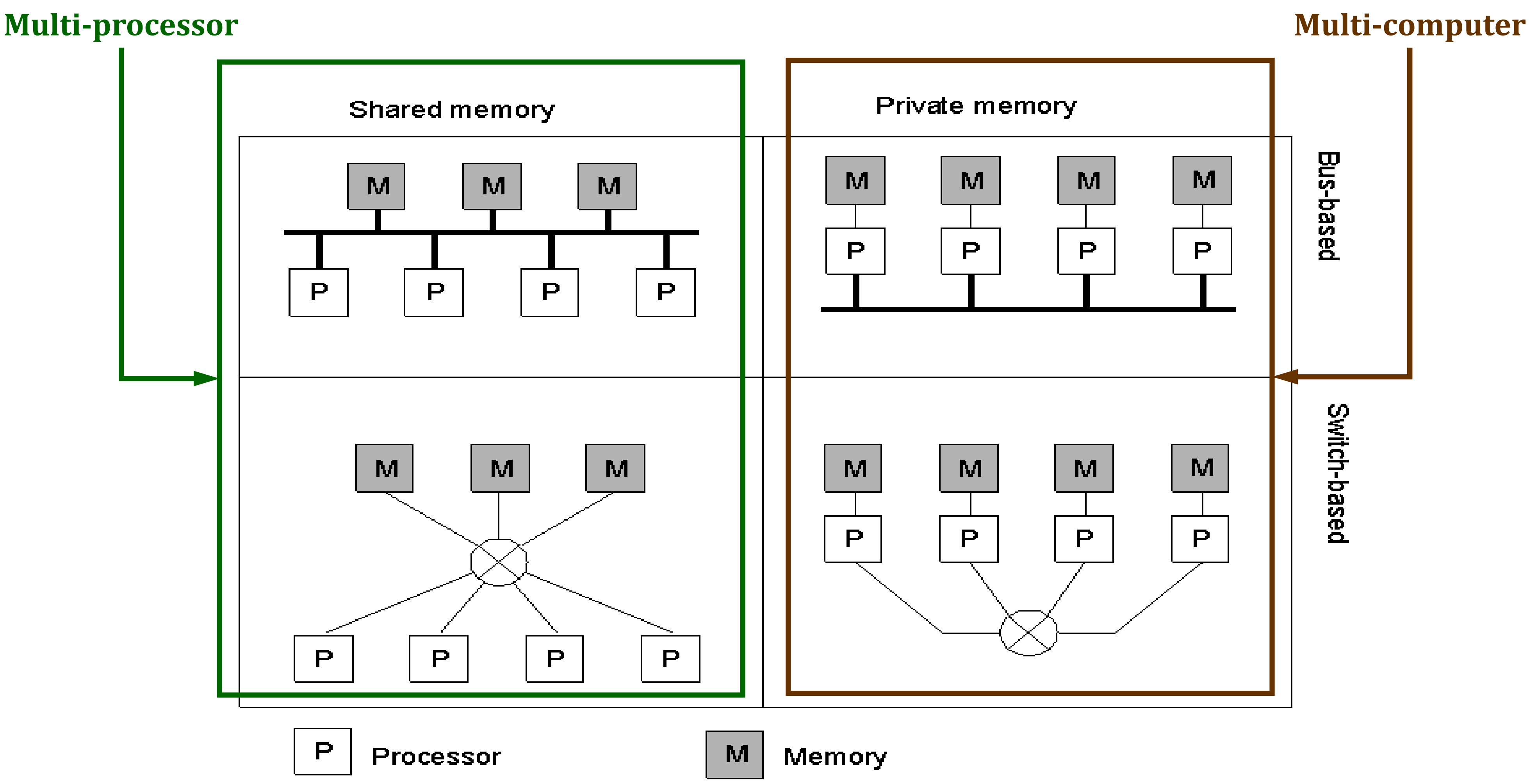
Prerequisites of distribution – 1

- An algorithm is distributed if
  - Every part of it acts satisfactorily on the basis of local knowledge
    - The DNS is partitioned
    - Blockchain is trustworthily replicated
  - Its computation does not require knowledge of global status
    - Local responses contribute to global result (DNS)
    - Local responses have global effect if confirmed by peers (Blockchain)
  - Local faults do not cause global failure
  - Its logic does not require a single source of time
  - It allows consistent replication of services, decomposition of tasks, partitioning of resources

Prerequisites of distribution – 2

- Synchronous communication is an active obstacle to distribution
  - It blocks the communicating parties delaying the progress of computation and causing coupling
- Asynchronous communication enables distribution
  - It decouples the communicating parties by hiding network delays, and allows parties to progress independently

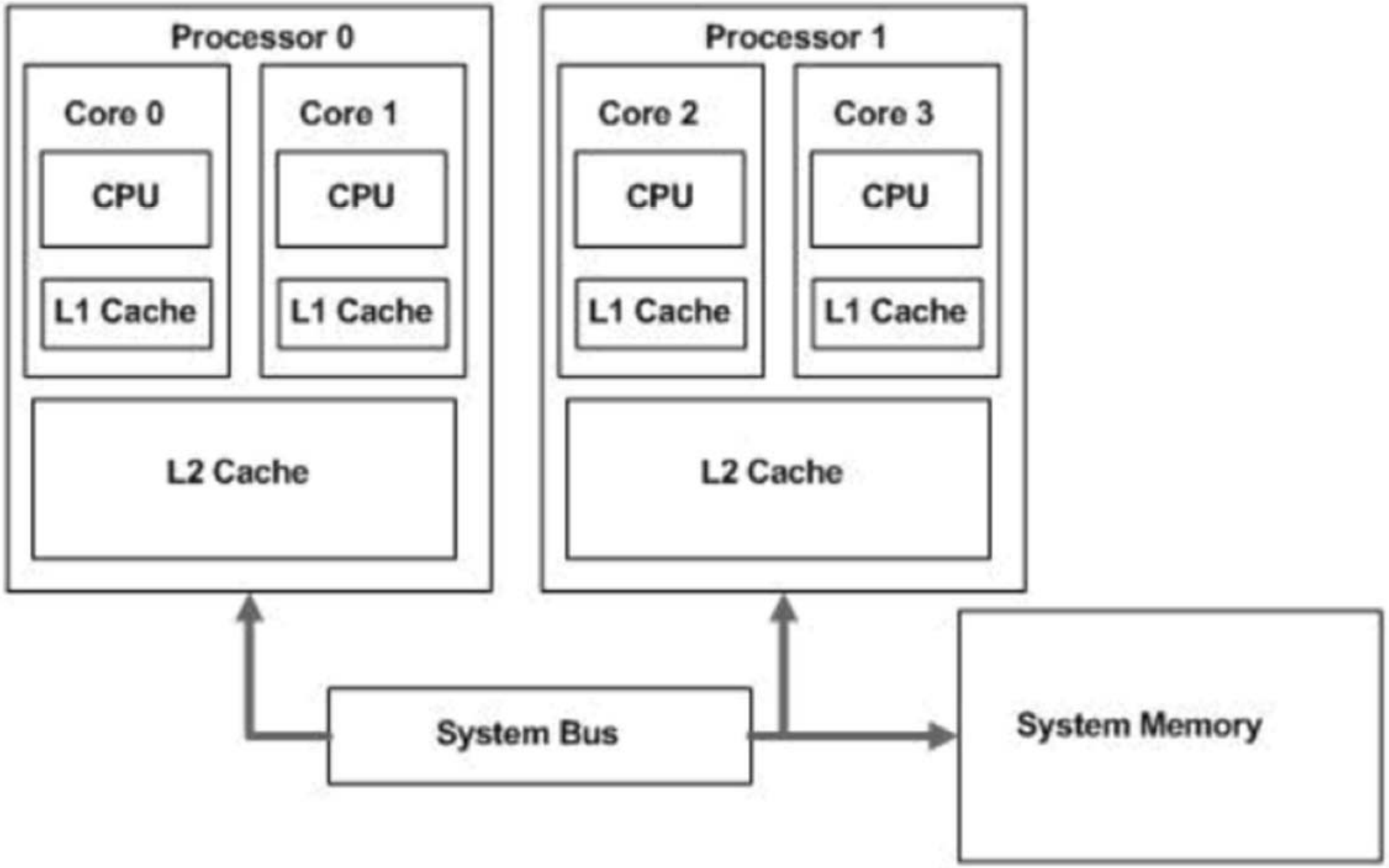
Hardware distribution



Distributed memory architecture

- Uniform memory access (UMA)
  - A single address space
    - As in symmetric multiprocessors
  - All node access memory in the same way
    - Access requests need queuing and arbitration
  - Cache coherence is not obvious
- Not-uniform memory access (NUMA)
  - Address space is shared but not unified
  - Access to memory depends on location
  - Cache coherence is unthinkable

Cache coherence – 1



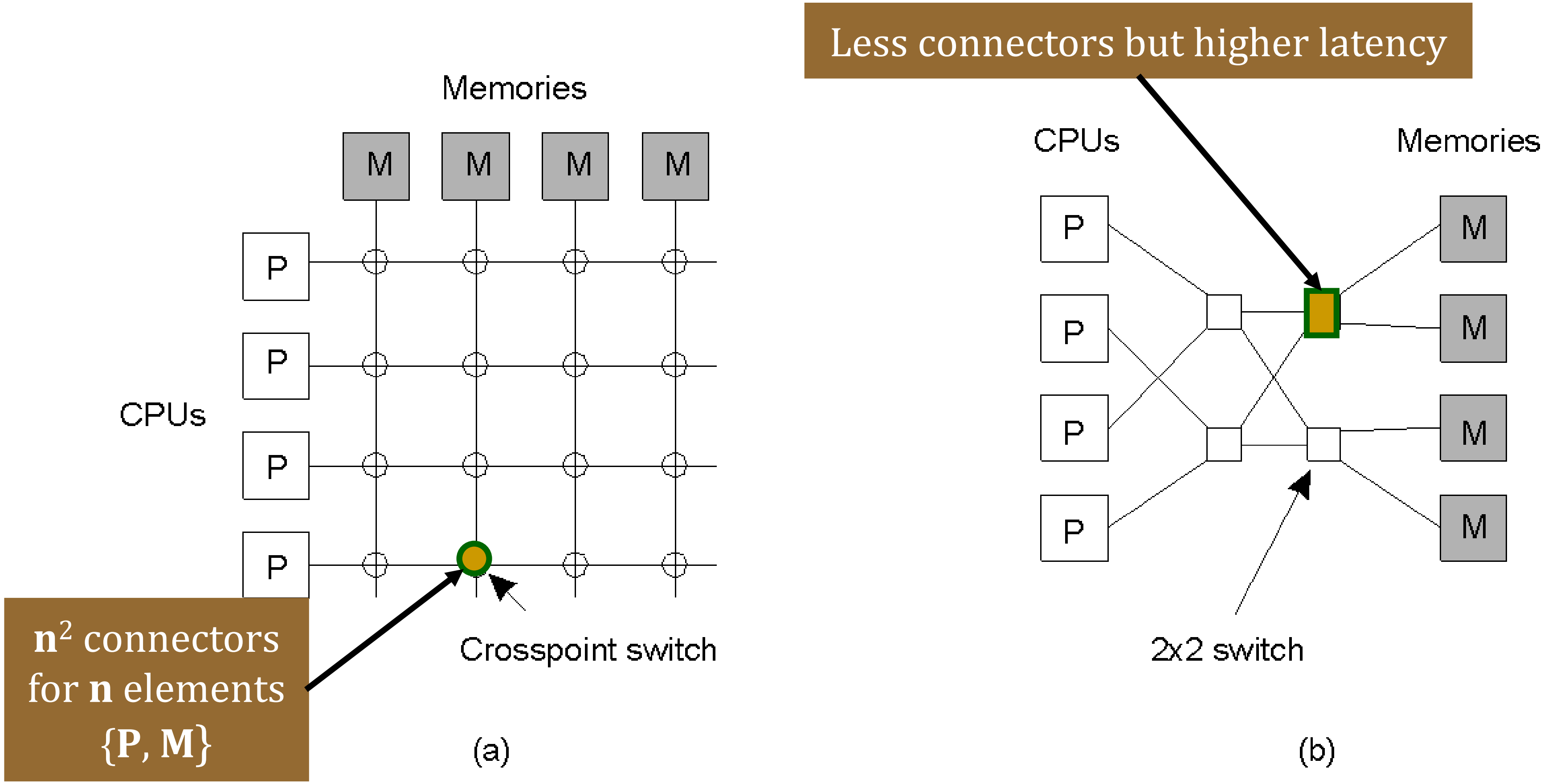
Cache coherence – 2

- The problem lies in cores having a private L1
  - Parallel R/W ops on same physical location see different values
- No-go remedies
  - Doing without caches kills performance
    - Nah, unless you want to kill performance
  - Sharing L1 across cores requires centralized arbitration
  - Write-through caches cause Rs to fail to see Ws from other cores
- Requirements
  - Every R must see the effect of every W
    - Either **write-update** or **write-invalidate**
  - Every R must see one and the same order of Ws
    - **Snooping**: order of Ws is determined by propagation on memory bus

Multiprocessors – 1

- All processors have a single common address space
  - Bus-based P-M communication requires arbitration and becomes a bottleneck
  - Switched P-M communication balances load better but requires far more complex logic
    - Crossbars are efficient but costly
    - Omega networks have cheaper units but are more complex to operate

Multiprocessors – 2

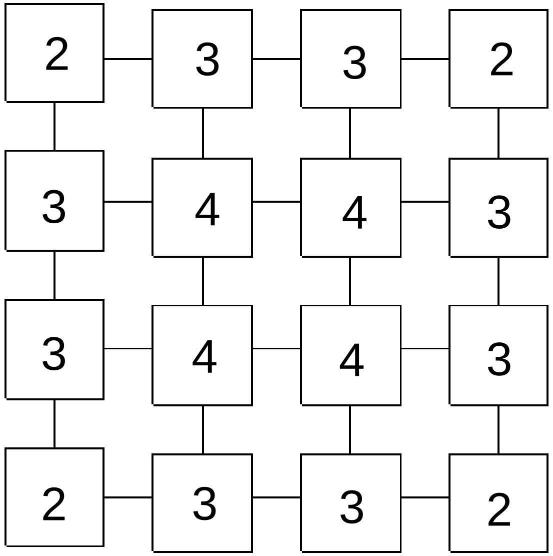


Crossbar switch **Omega network**

Multi-computers

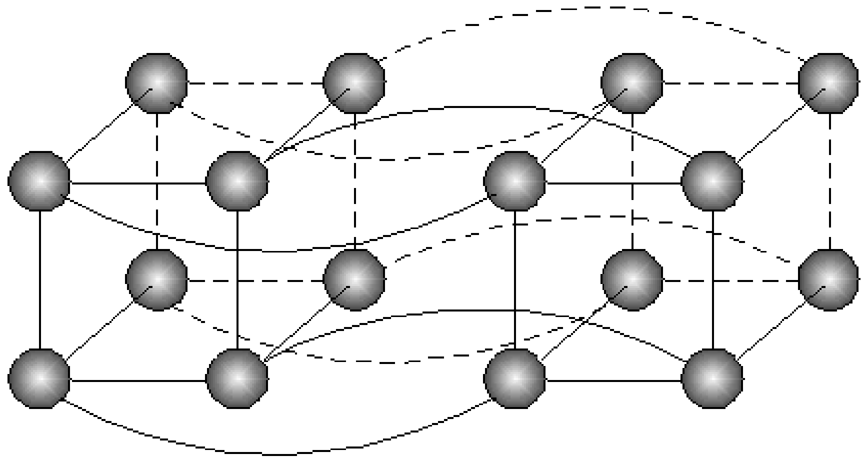
Every node does local processing and routing

2^n nodes
$n2^{n-1}$ links



Grid

Node position determines
number of neighbours
(position-dependent routing)

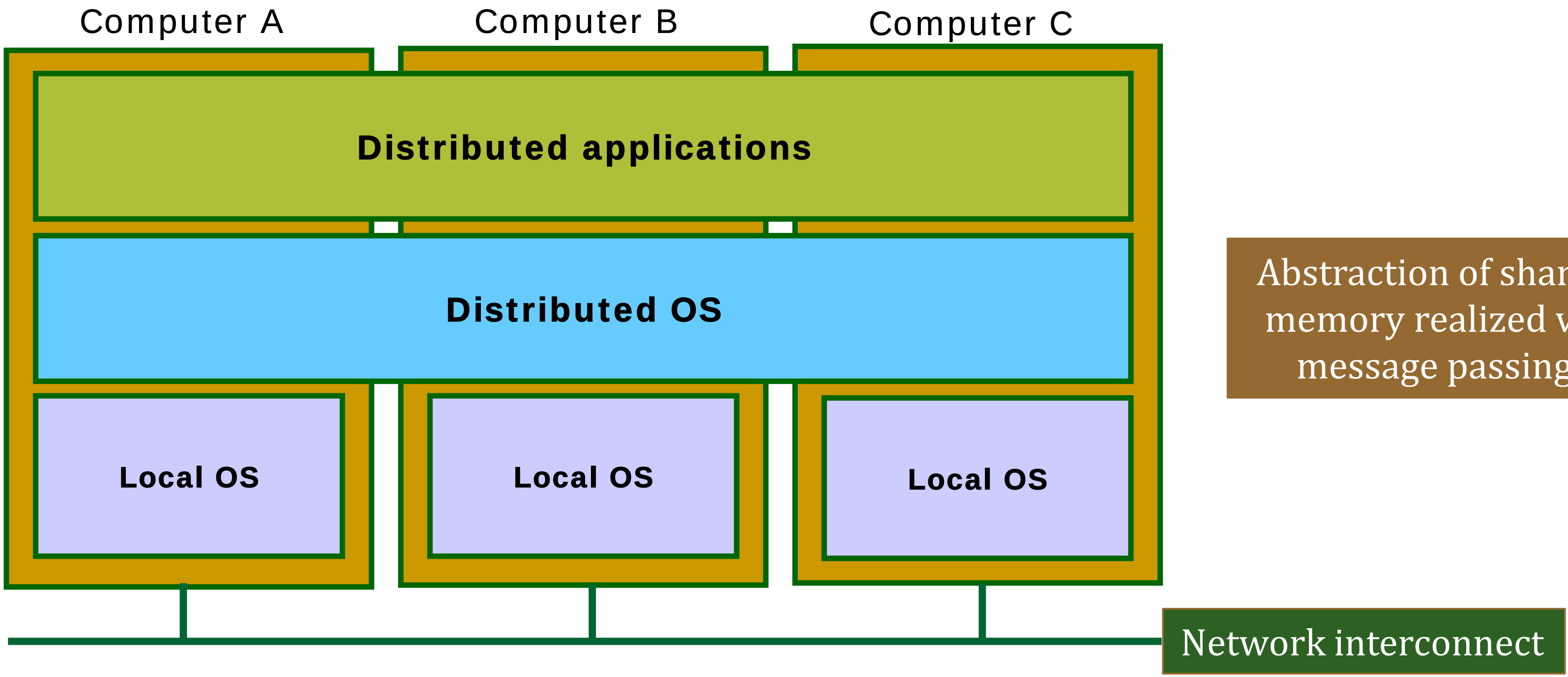


n = 4

Hypercube

Number of neighbours is location independent
(and so is routing)

Software distribution – 1



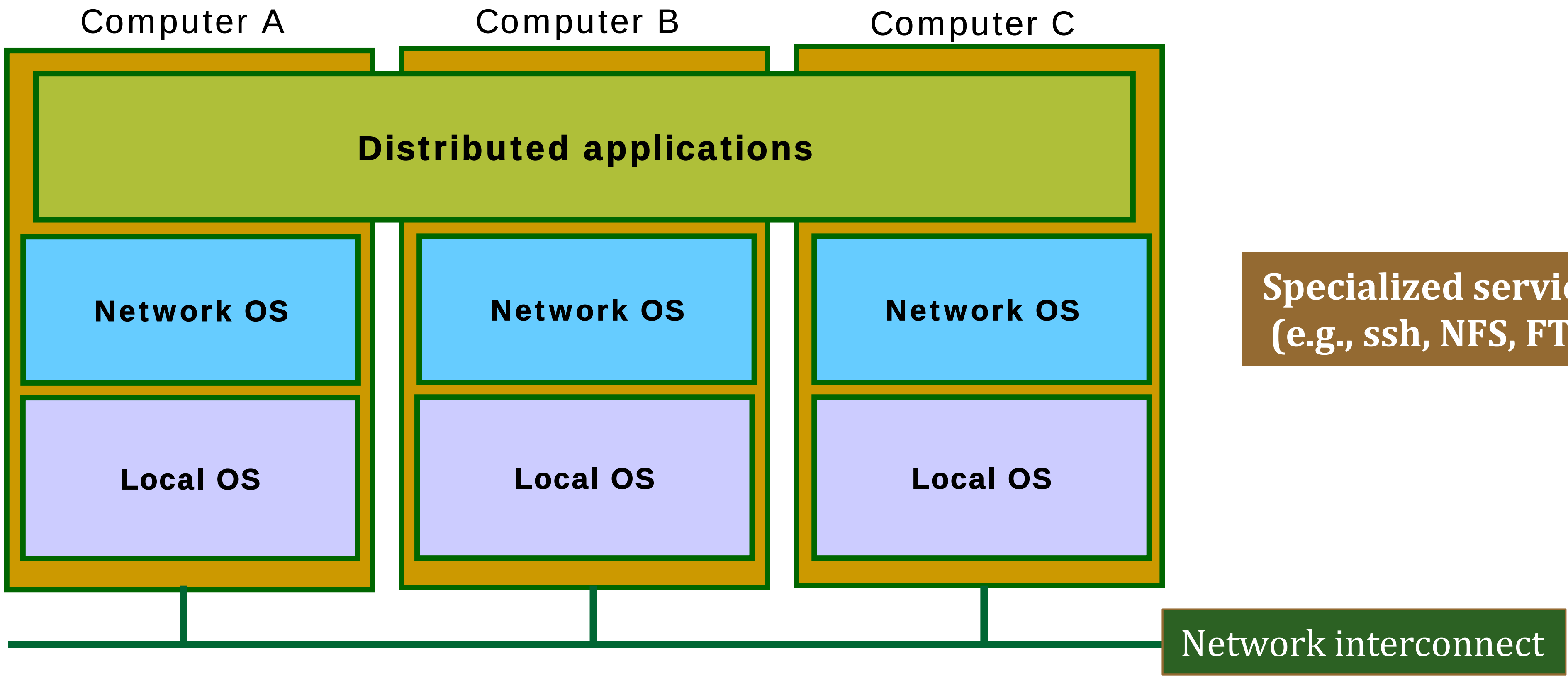
Abstraction of shared memory realized via message passing

Typically intended for homogeneous systems

Software distribution – 2

- Programming distributed systems is harder than doing so for multiprocessors
  - Task scheduling is much harder in the latter
  - Resource sharing is complex in the former and may prefer spin locks to suspend locks in the latter
- Communicating by shared memory is simpler than by message passing
  - The former is natural in multiprocessors
  - The latter scales nicely but suffers from queuing, synchronization, coordination, and network effects

Software distribution – 3



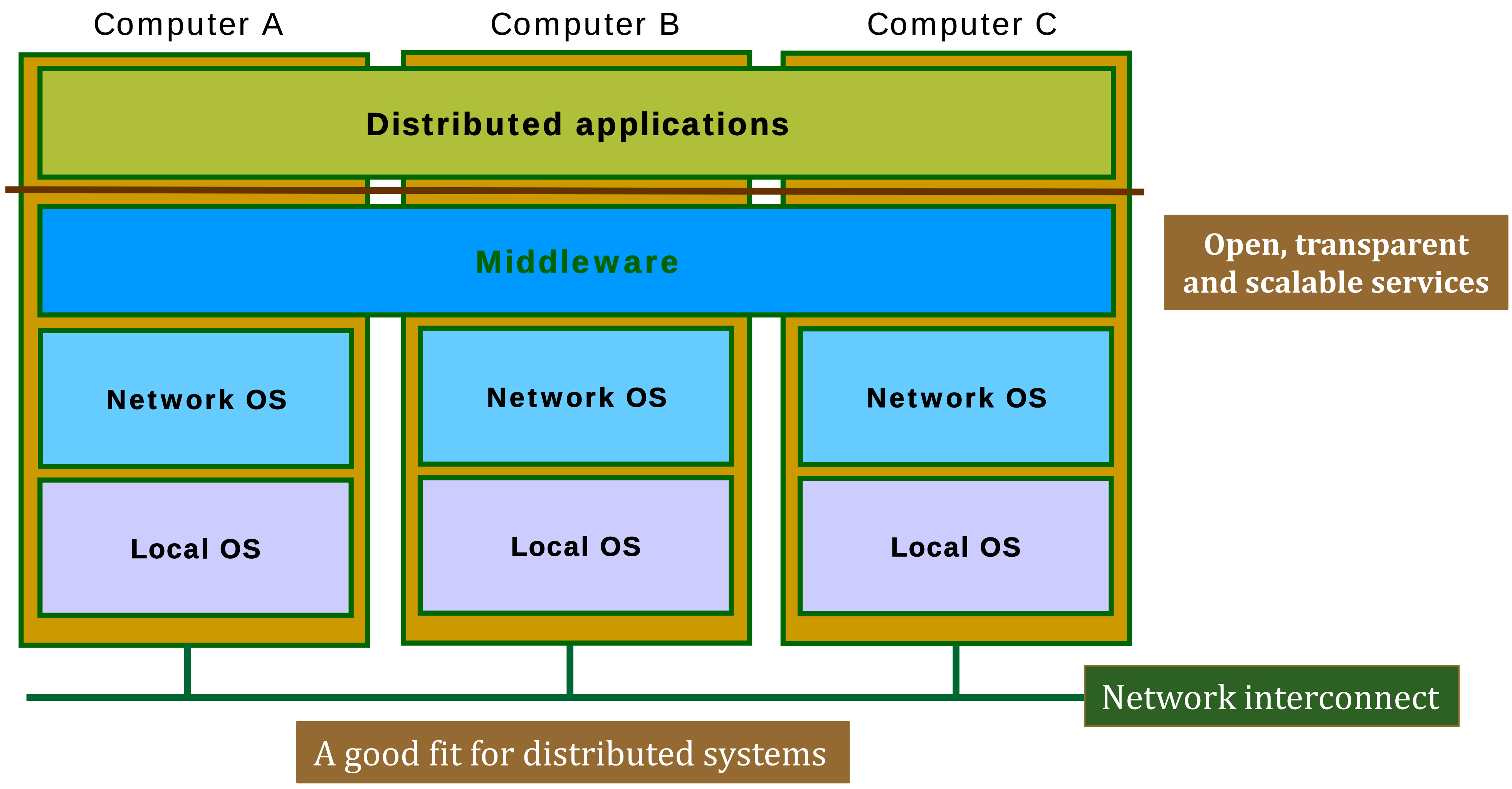
Specialized services (e.g., ssh, NFS, FTP)

Typically intended for heterogeneous systems

Software distribution – 4

- Neither the distributed OS nor the network OS paradigm conform with the definition of distributed system
  - The former may have good transparency but its participant nodes are **not** independent
  - The latter may have good openness and scalability features but it does **not** yield a united coherence system
- The new means to software distribution is called **middleware**

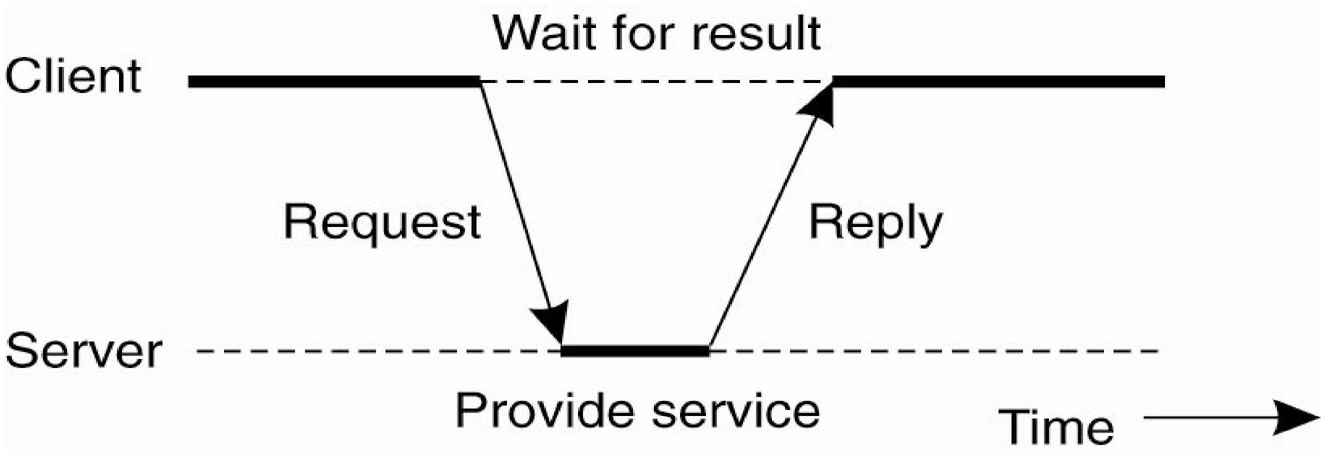
Software distribution – 5



Variants of middleware

- Distributed file system
  - UNIX-like NFS
- Remote procedure call (RPC)
- Distributed objects (RMI)
- Distributed documents: Web 1.0
  - All TCP based
- Distributed everything: Web 2.0 (all over HTTP)
  - Resource-centric: REST
  - Data-centric: GraphQL
  - Collaboration-centric: gRPC
  - Stream-oriented: WebRTC

Styles of distributed interaction – 1

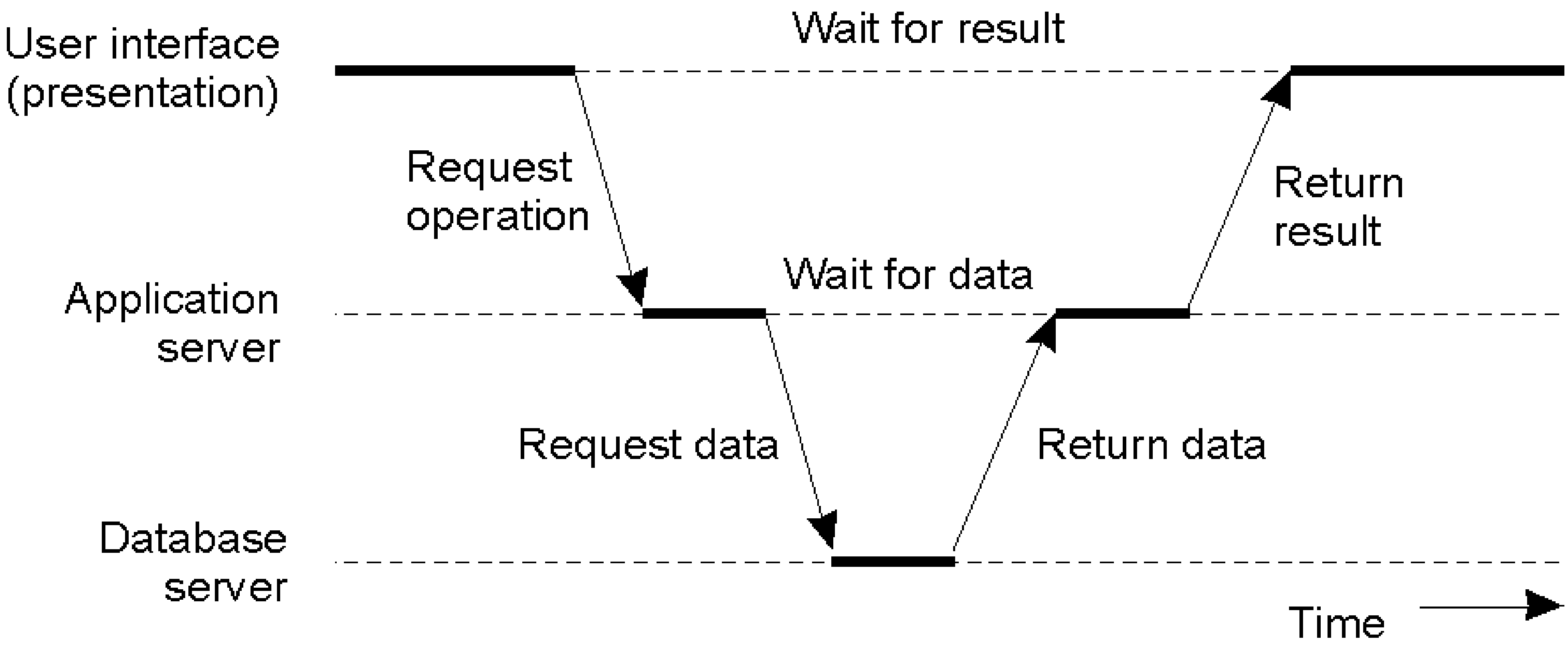


- The **request-reply** style of interaction was the killer factor in the Web 1.0 world
  - Reissuing requests in the absence of replies is harmless only for **idempotent** operations
  - Very few operations are so …

Styles of distributed interaction – 2

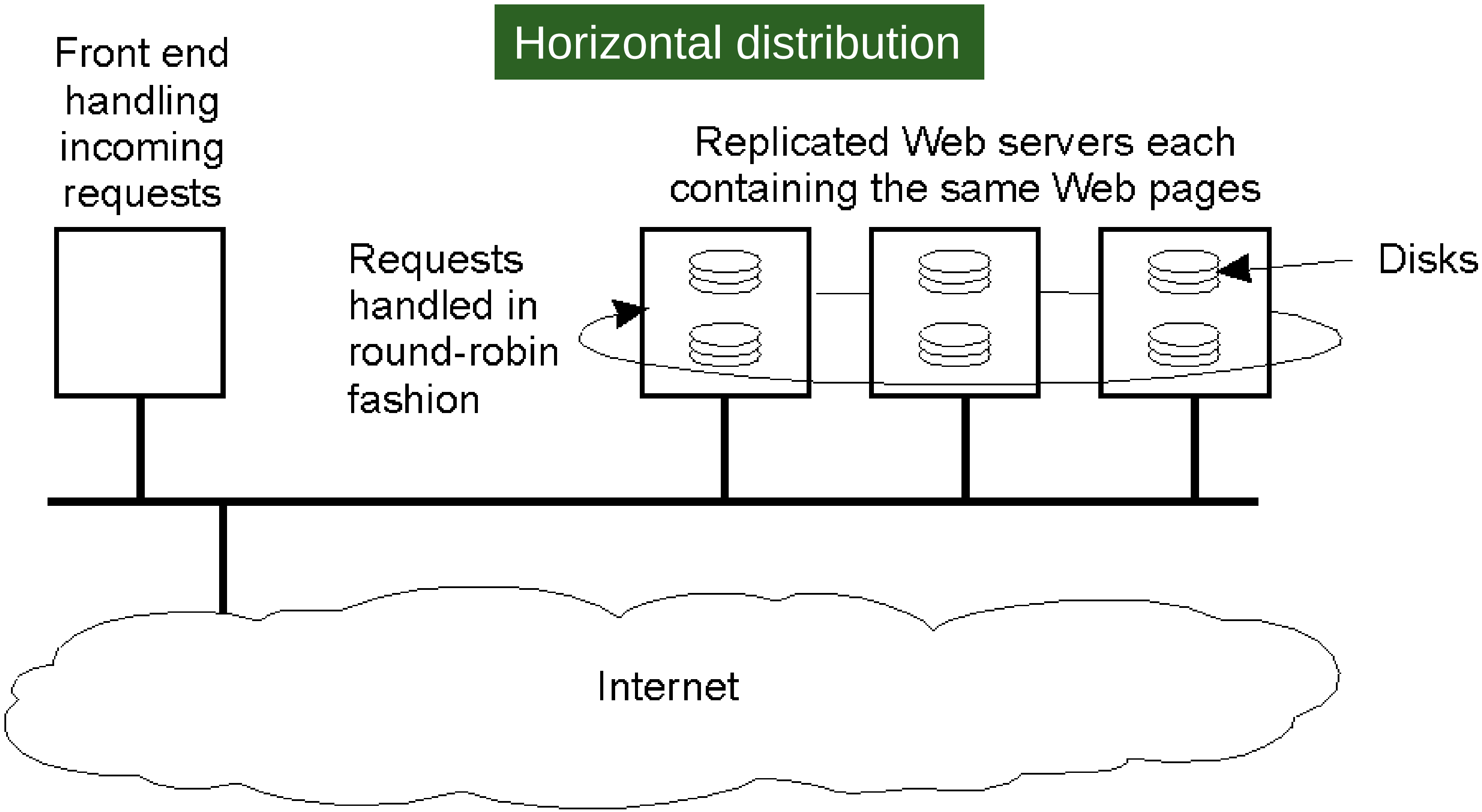
- Client-server architectures vary according to the distribution of either service or data
- Distribution is **vertical** when service is decomposed across multiple authorities
  - Akin to functional pipelining: specialization
  - Overall service needs coordination of parts
- Distribution is **horizontal** when data is replicated across multiple identical servers
  - Fit for load balancing
  - Consistency must be preserved across replicas

Styles of distributed interaction – 3



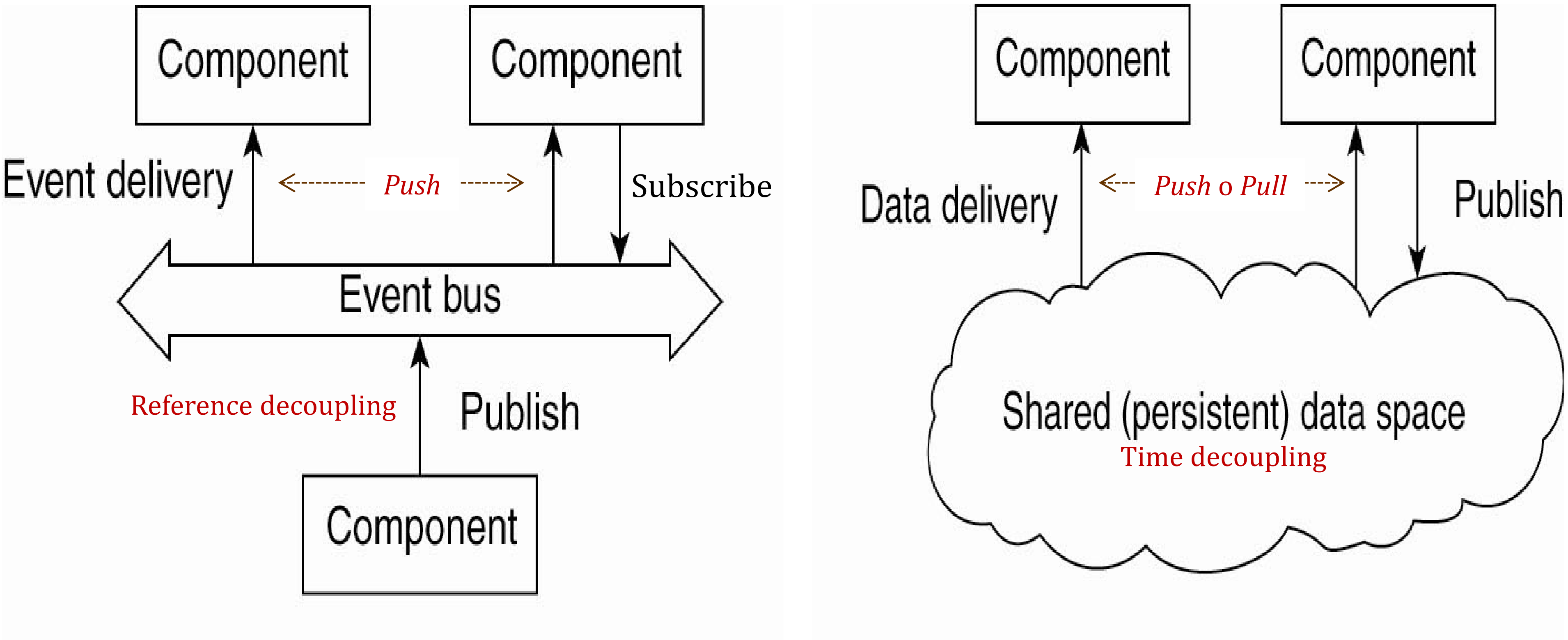
Vertical distribution

Styles of distributed interaction – 4



Styles of distributed interaction – 5

Beyond client-server



Tanenbaum & Van Steen, *Distributed Systems: Principles and Paradigms*, 2e, (c) 2007 Prentice-Hall, Inc.

Views of a remote call

Tanenbaum & Van Steen, *Distributed Systems: Principles and Paradigms*, 2e, (c) 2007 Prentice-Hall, Inc.