Fifth SimLab Short Course on

Parallel Numerical Simulation

Belgrade, October 1—7, 2006

Message-Coupled Systems

October 3, 2006

Dr. Ralf-Peter Mundani
Department of Computer Science – Chair V
Technische Universität München, Germany

What’s Left…

- problem: simulation of physical phenomenon/technical process
  - mathematical model ✓
  - discretisation ✓
  - algorithm development ✓
  - implementation ✓
  - → running (sequential) code

- what’s left: parallelisation

What’s Left…

- problem: simulation of physical phenomenon/technical process
  - mathematical model ✓
  - discretisation ✓
  - algorithm development ✓
  - implementation ✓

  → running (sequential) code

- what’s left: parallelisation
  - “…now you **only** have to do **some** parallelisation…”

  ? Question: how much time does one need for this

Examples of Parallel Prog. Languages

- Occam
  - imperative procedural language
  - builds on Communicating Sequential Processes formalism
- Linda
  - basically four operations: in, rd, out, eval
  - tupels can be added, retrieved, or destructively retrieved from logical associative memory (tuplespace)
  - extension of other languages such as Prolog, C, or Java
- OpenMP
  - set of compiler directives for shared memory architectures
  - work load distribution (work sharing) using threads
  - simple to program (no dramatic change to code needed)

The Message Passing Paradigm

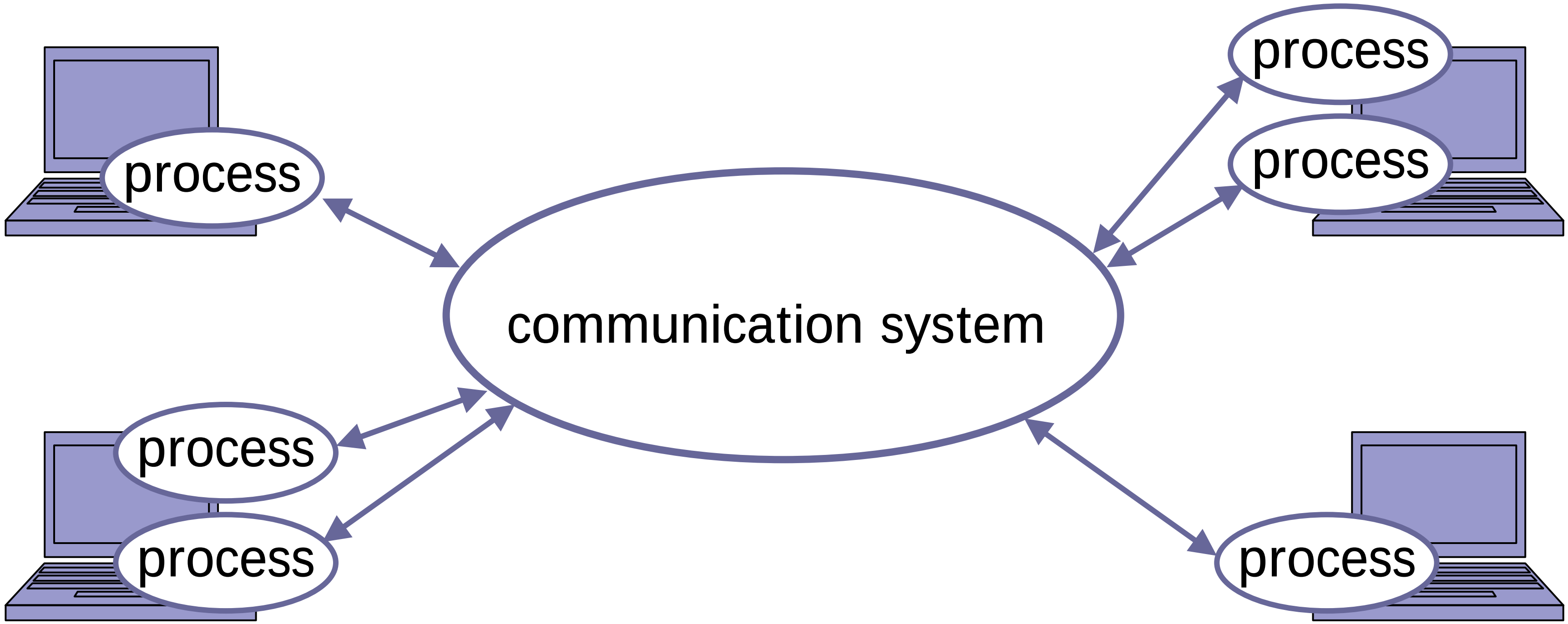
- very general principle, applicable to nearly all types of parallel architectures (message-coupled and memory-coupled)
- standard programming paradigm for message-coupled systems
  - message-coupled multiprocessors
  - cluster of workstations (homogeneous, dedicated use, high-speed network)
  - networks of workstations (heterogeneous, non-dedicated use, standard network (e.g. ethernet))
- several concrete programming environments
  - machine-dependent: MPL (IBM), PSE (nCUBE), …
  - machine-independent: EXPRESS, P4, PARMACS, PVM, …
- machine-independent standards: PVM, MPI

The Underlying Principle

- parallel program with *p* processes with different address space
- communication takes place via exchanging messages
  - header: target ID, message information (type of data, …)
  - body: data to be provided
- exchanging messages via library functions that should be
  - designed without dependencies of
    - hardware
    - programming language
  - available for multiprocessors and standard monoprocessors
  - available for standard languages such as C/C++ or Fortran
  - linked to source code during compilation

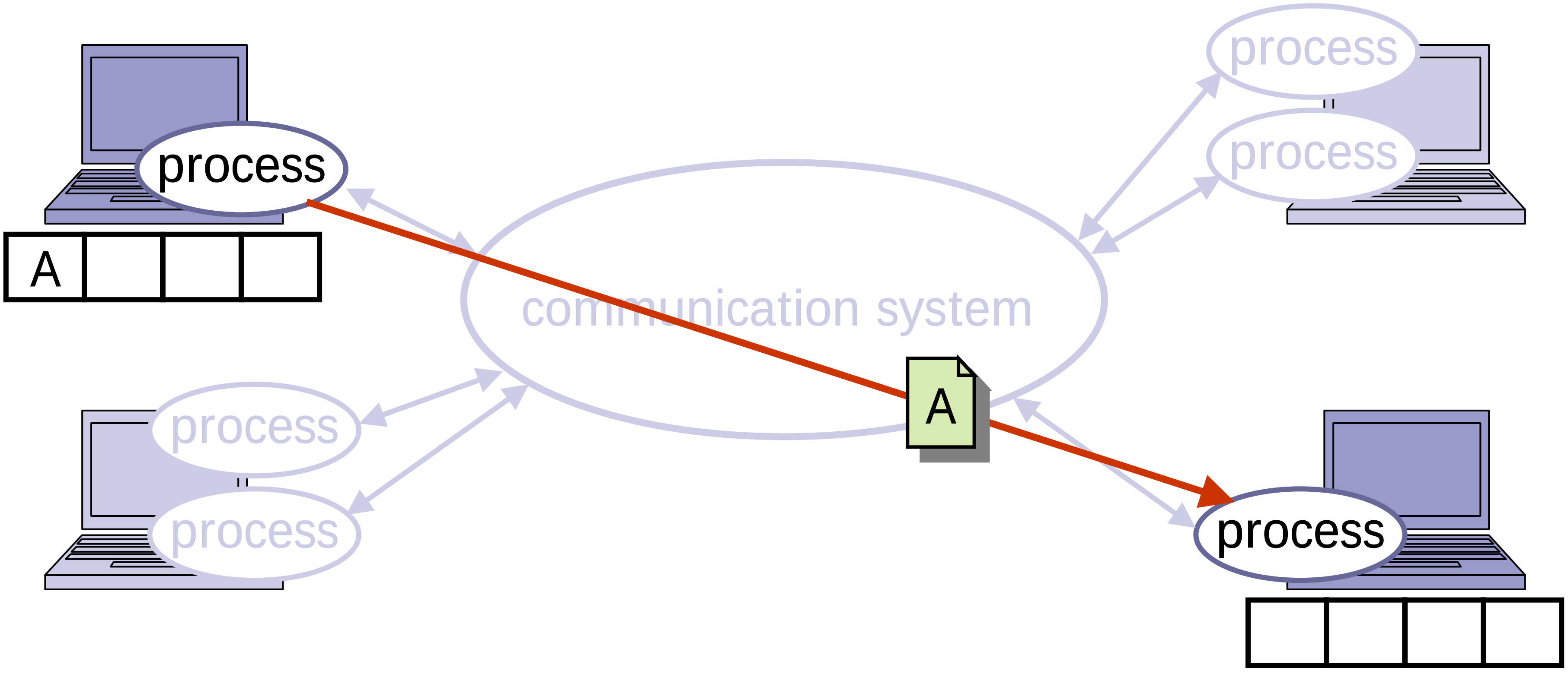
The User's View

- library functions are the only interface to communication system



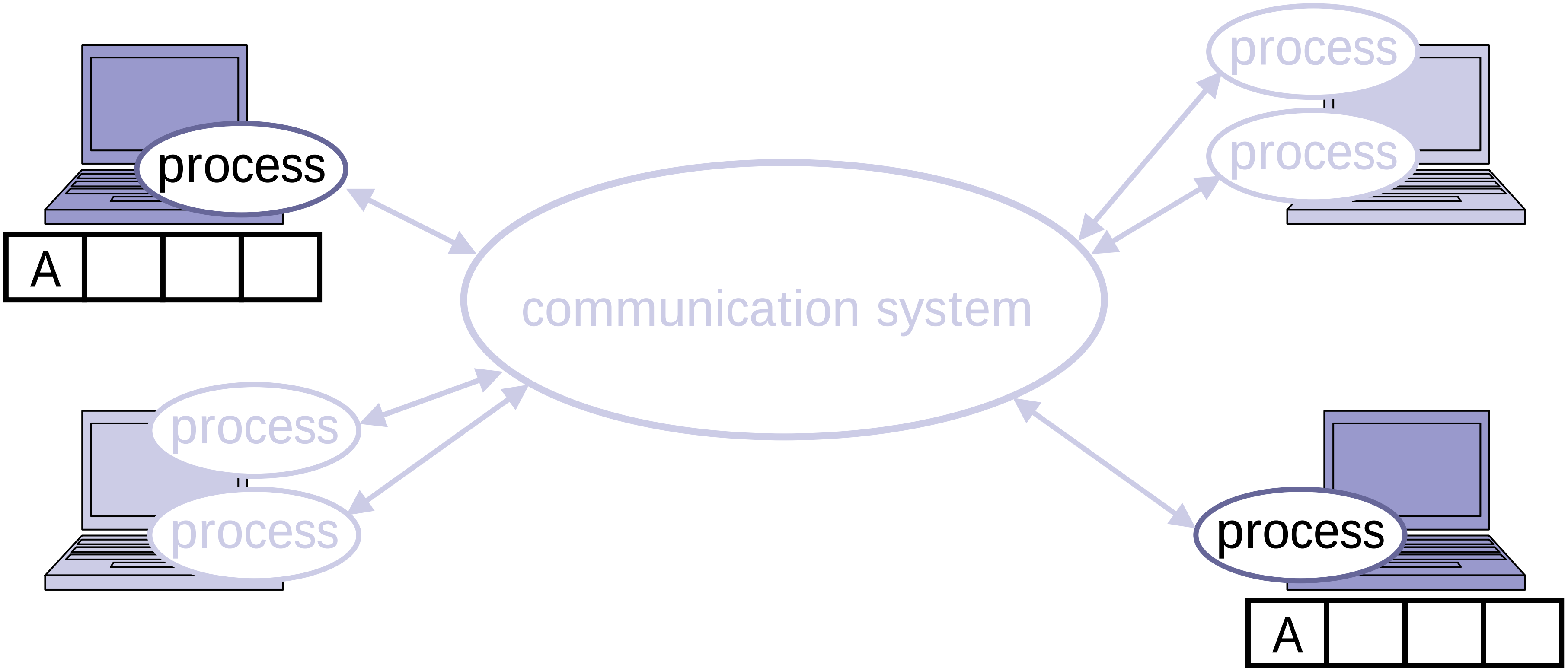
The User's View

- library functions are the only interface to communication system
- message exchange via send() and receive()



The User's View

- library functions are the only interface to communication system
- message exchange via send() and receive()



Elementary Communication

- point-to-point (1:1-communication)
- collective (1:*m*-communication, $m \leq n$, n number of processes)
- communication operations
  - send
    - required: receiver, send buffer, type of message, communication context
    - blocking: continuation possible after passing message to communication system has been completed (thus, buffer can be re-used)
    - non-blocking: immediate continuation possible; further test whether message has been sent and buffer can be re-used necessary

Elementary Communication

- communication operations (cont'd)
  - receive
    - required: sender (wildcards possible), receive buffer, type of message, communication context
    - blocking: continuation only after (suitable) message has been received
    - non-blocking: immediate continuation possible, independent from result (success/failure); further test whether message has been arrived and buffer can be re-used necessary

Message Buffers

- typically (but not necessarily) connected parts of memory
  - homogeneous architectures: sequence of bytes
  - heteregeneous architectures: type information necessary for format conversion by message passing library (e.g. size of datatypes, byte order)
- definition and allocation of message buffers
  - send buffer: generally done by application program
  - receive buffer: either automatically by message passing library or manually by application program

Message Buffers

- why buffers?

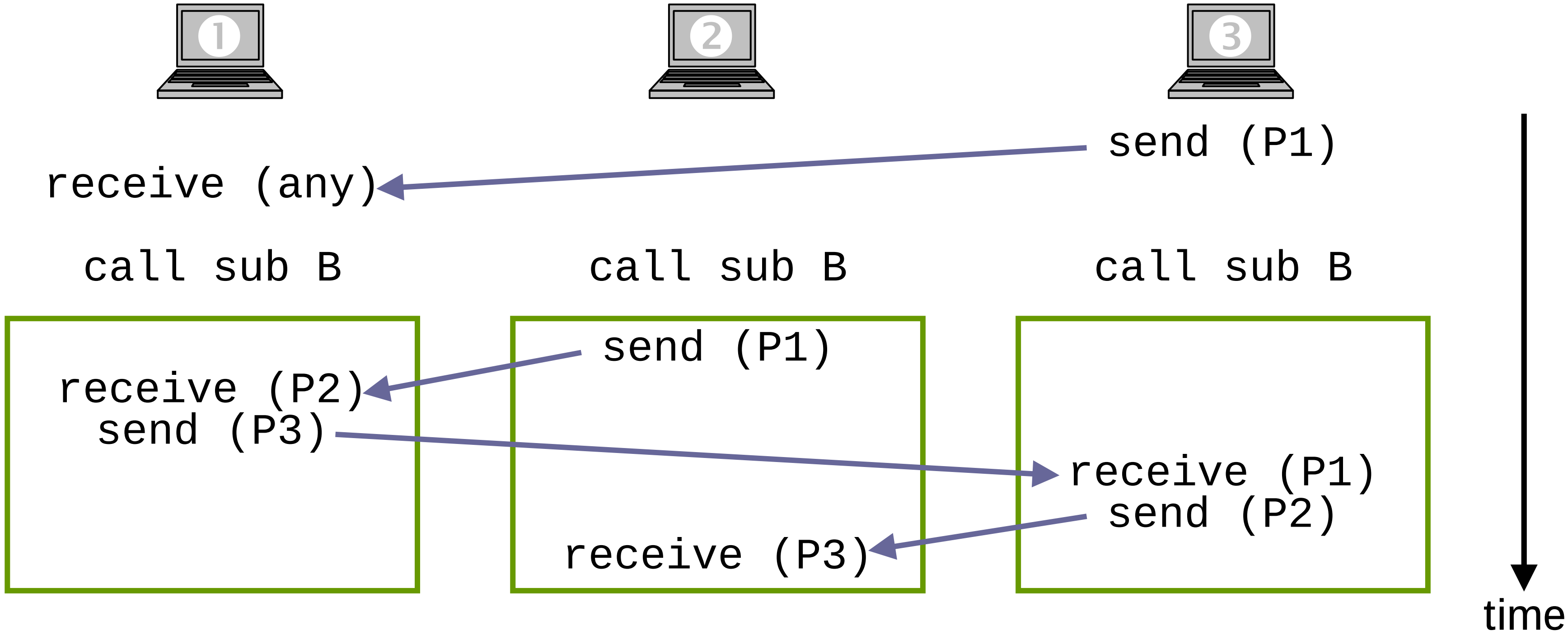
```
P1: compute something
    store result in SBUF
    SendBlocking(P2, SBUF)
    RecvBlocking(P2, RBUF)
    read data in RBUF
    process RBUF
```

```
P2: compute something
    store results in SBUF
    SendBlocking(P1, SBUF)
    RecvBlocking(P1, RBUF)
    read data in RBUF
    process RBUF
```

- does this work?
  - yes, if communication system buffers internally
  - no, otherwise (deadlock) – avoid via non-blocking communication or via atomic sendreceive operation

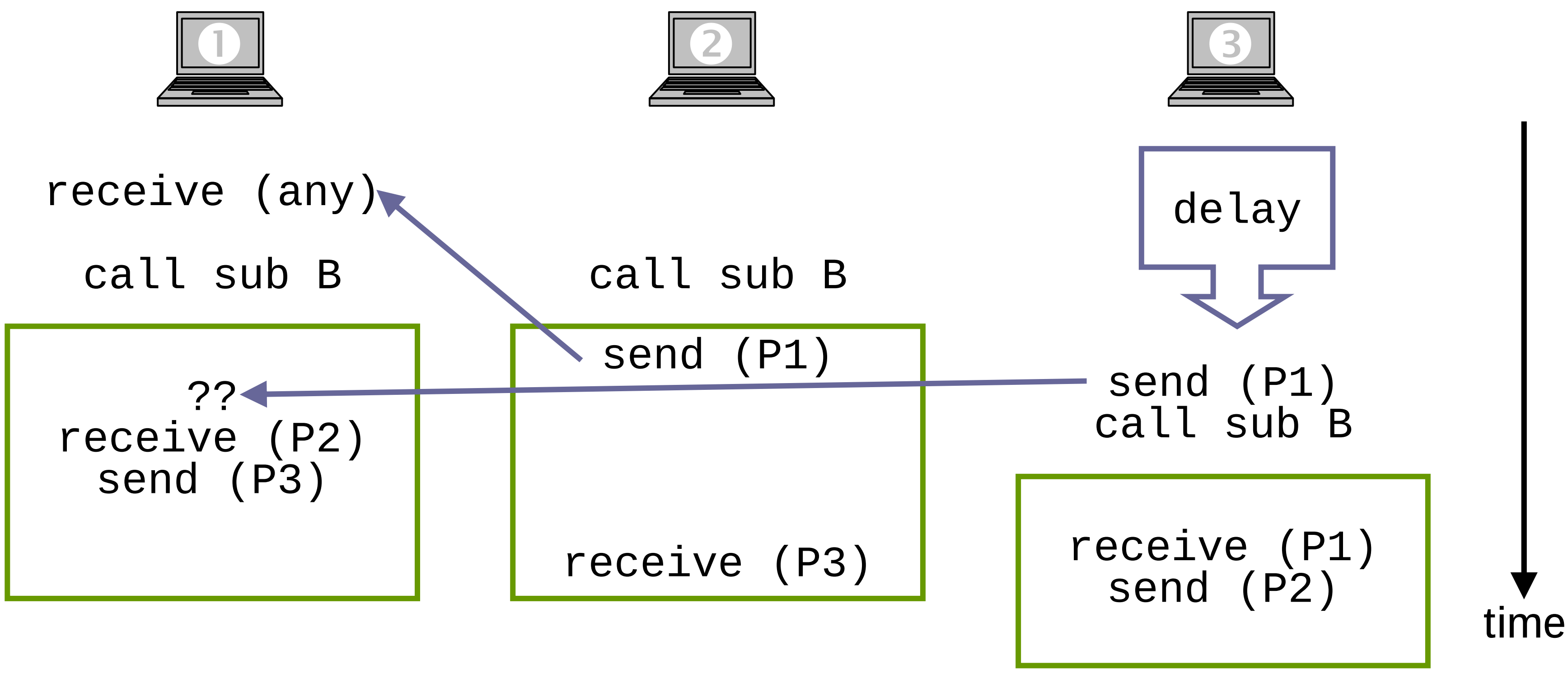
Communication Context

- three processes, all of them call subroutine B from a library
- inter-process communication within these subroutines
- communication context shall ensure this restriction to subroutines



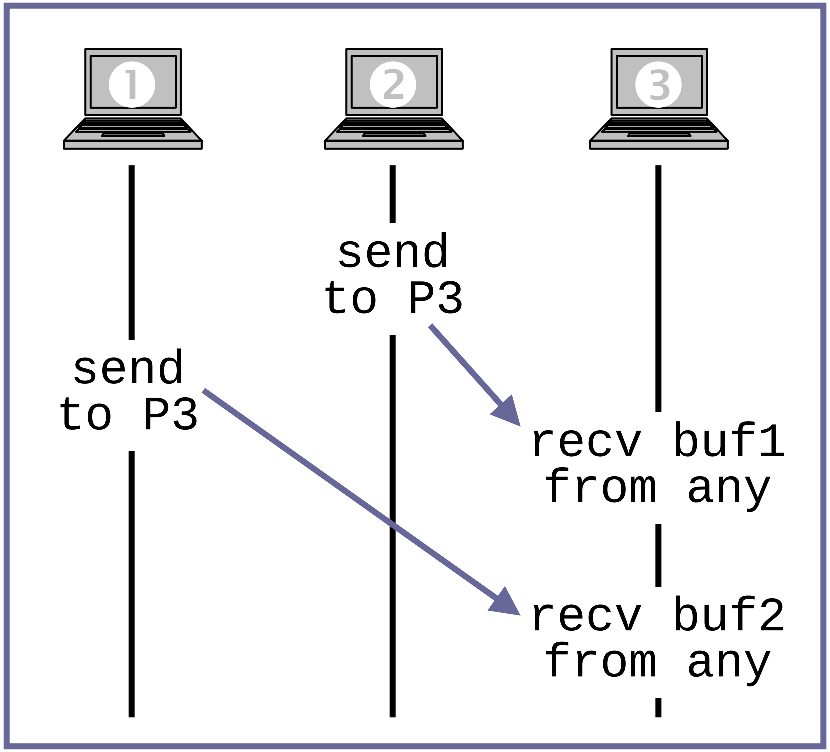
Communication Context

- three processes, all of them call subroutine B from a library
- inter-process communication within these subroutines
- communication context shall ensure this restriction to subroutines

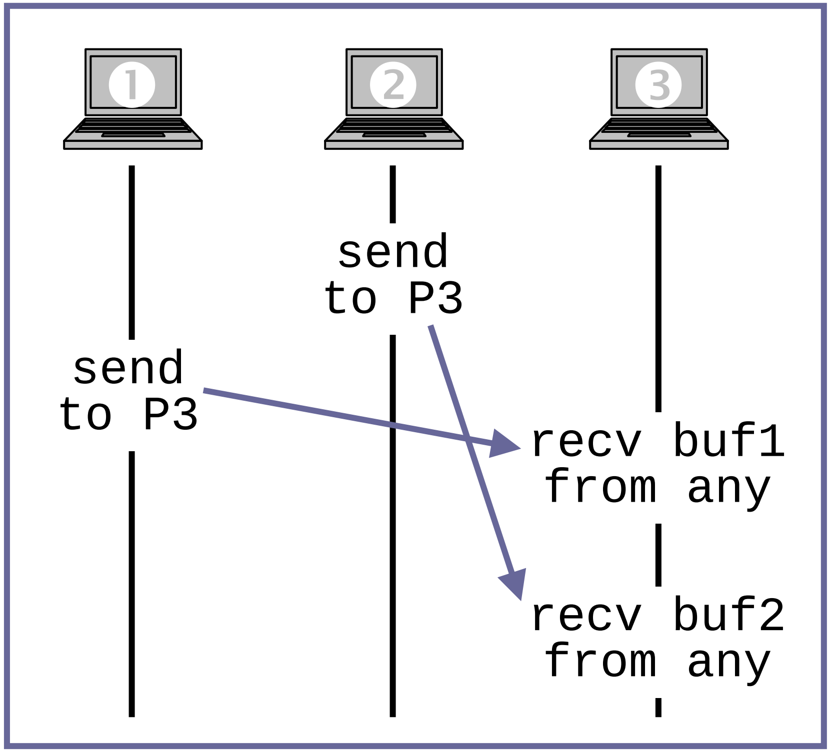


Keeping the Order

- problem: there is no global time in a distributed system
- consequence: maybe wrong send-receive assignments (for more than two processes and the usage of wildcards)



or



Message Types

- two main classes
  - data messages
    - data are exchanged for other processes' computations
    - example: border values of partial matrix in numerical solver
  - control messages
    - data are exchanged for other processes' control
    - example: competitive search for social security numbers in large data sets (e.g. 1.3 billion Chinese)
- in general, additional information about format necessary for both cases (provided along with message type)

Efficiency

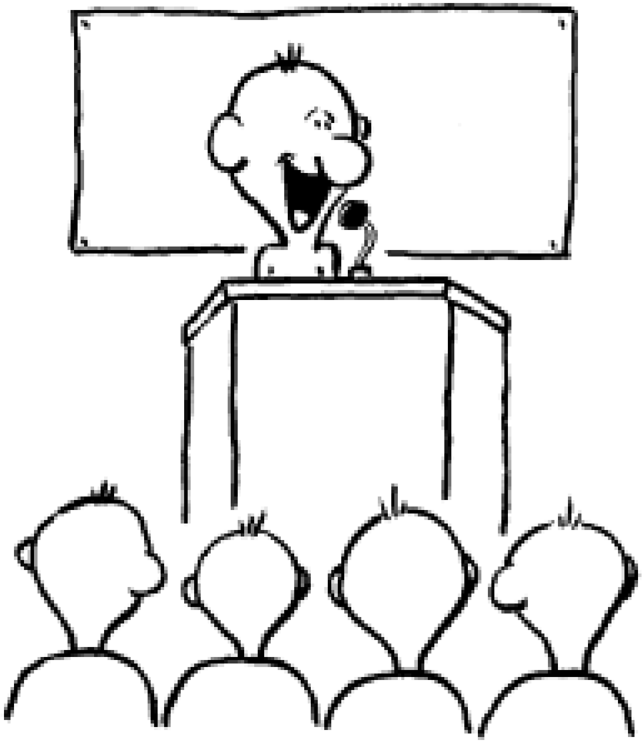
- avoid short messages: latency reduces the effective bandwith

  $t_{total} = t_{setup} + n/B$ with length *n*, bandwith *B*

  $B_{eff} = n/t_{total}$

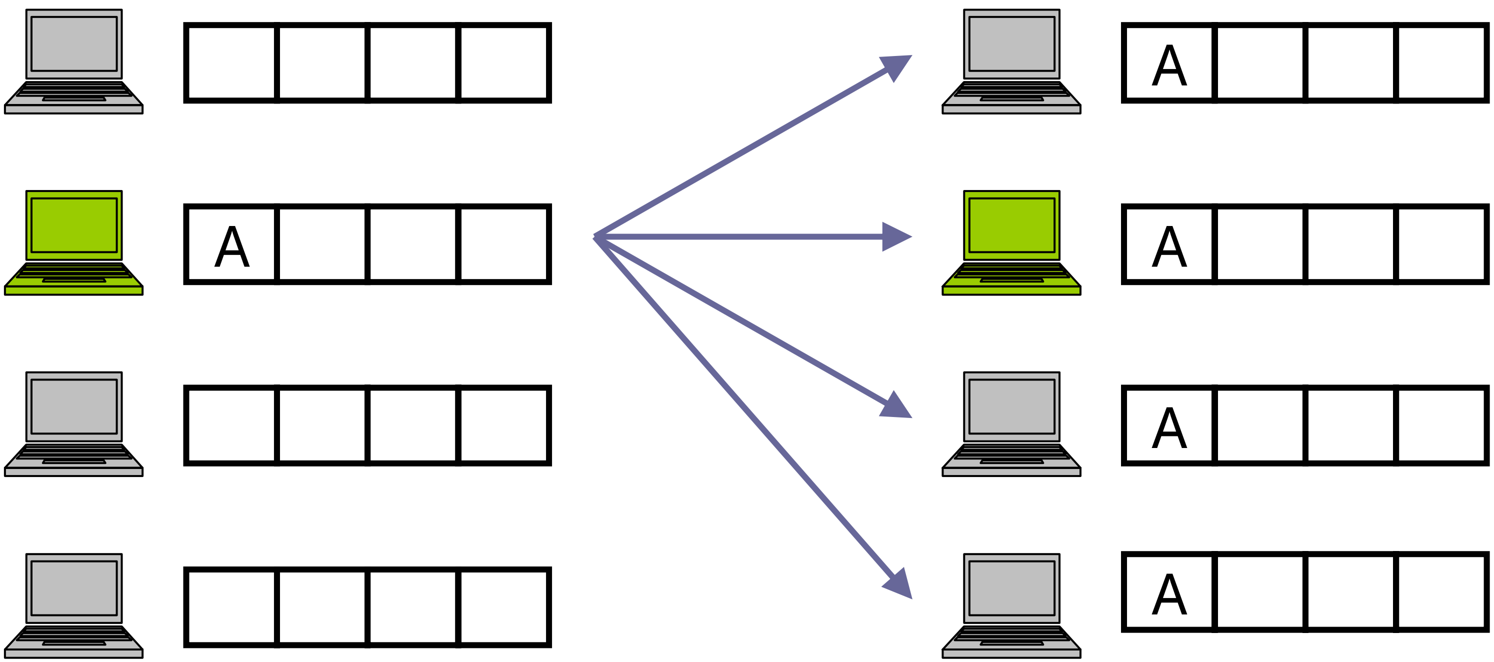
- computation should dominate communication
- typical conflict for numerical simulations
  - overall runtime suggests large number of processes
  - CCR and message size suggest small number of processes
- finding (machine- and problem-dependent) optimum number of processes
- try avoiding communication points at all, redundant computations prefered (if inevitable)

Collective Communication



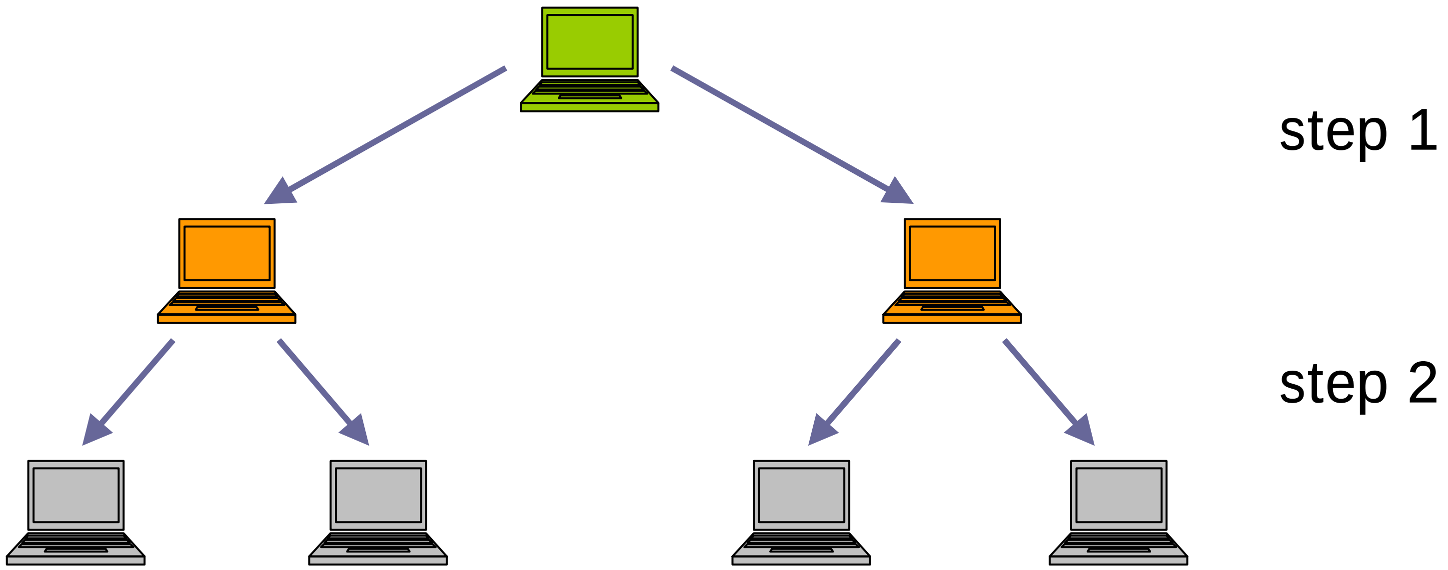
Collective Communication

- broadcast
  - sends message to all participating processes
  - example: first process in competition informs others to stop
  - ? Question: efficient implementation of a broadcast



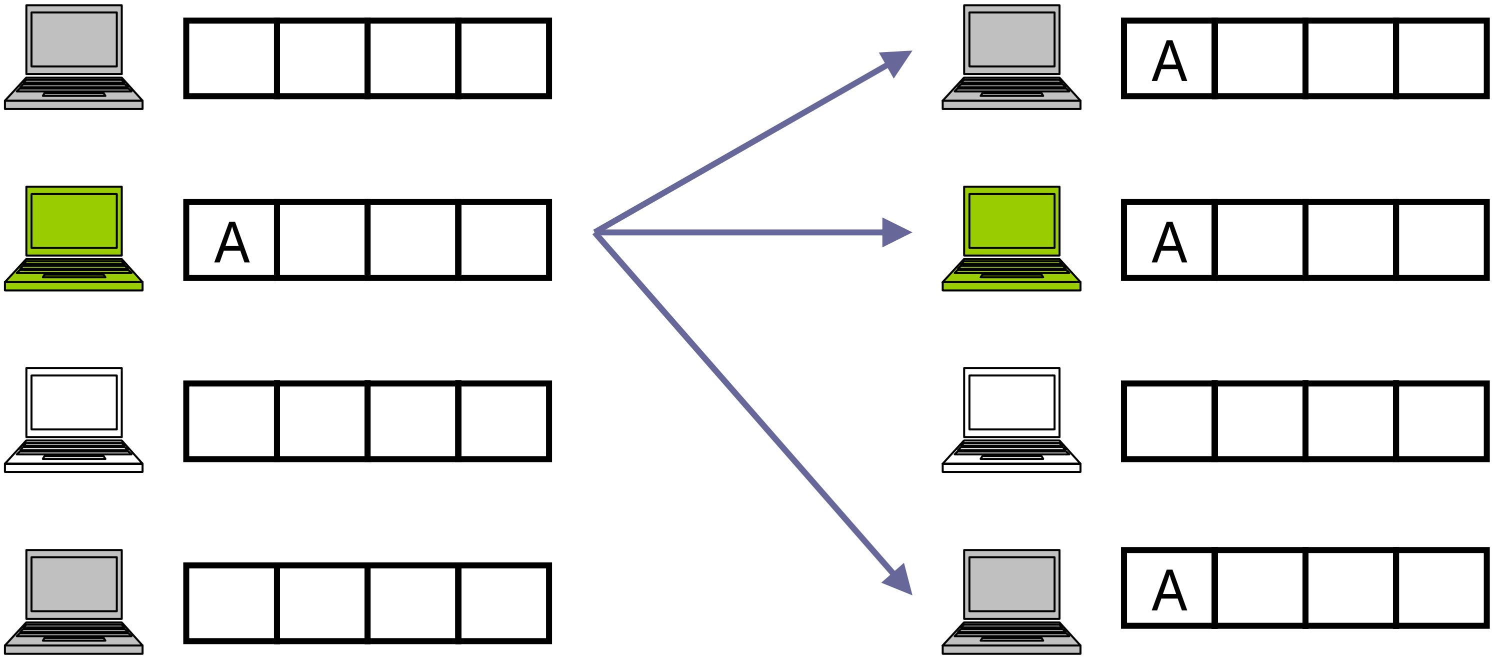
Collective Communication

- broadcast (cont'd)
  - using a binary tree
  - Question: how many steps for n processes



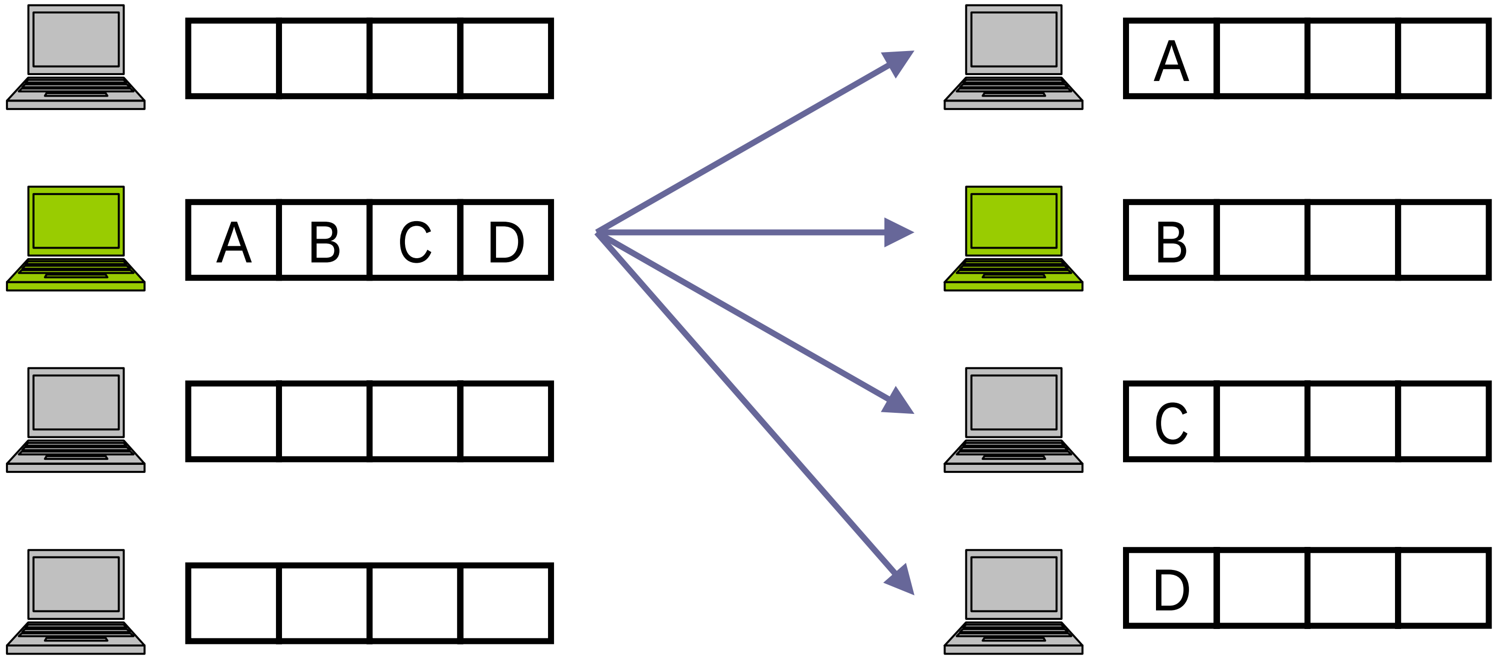
Collective Communication

- multicast
  - sends message to a subset of participating processes ($1:m$ communication with $m \leq n$; n number of processes)
  - example: update of (local) iterated solution to neighbours



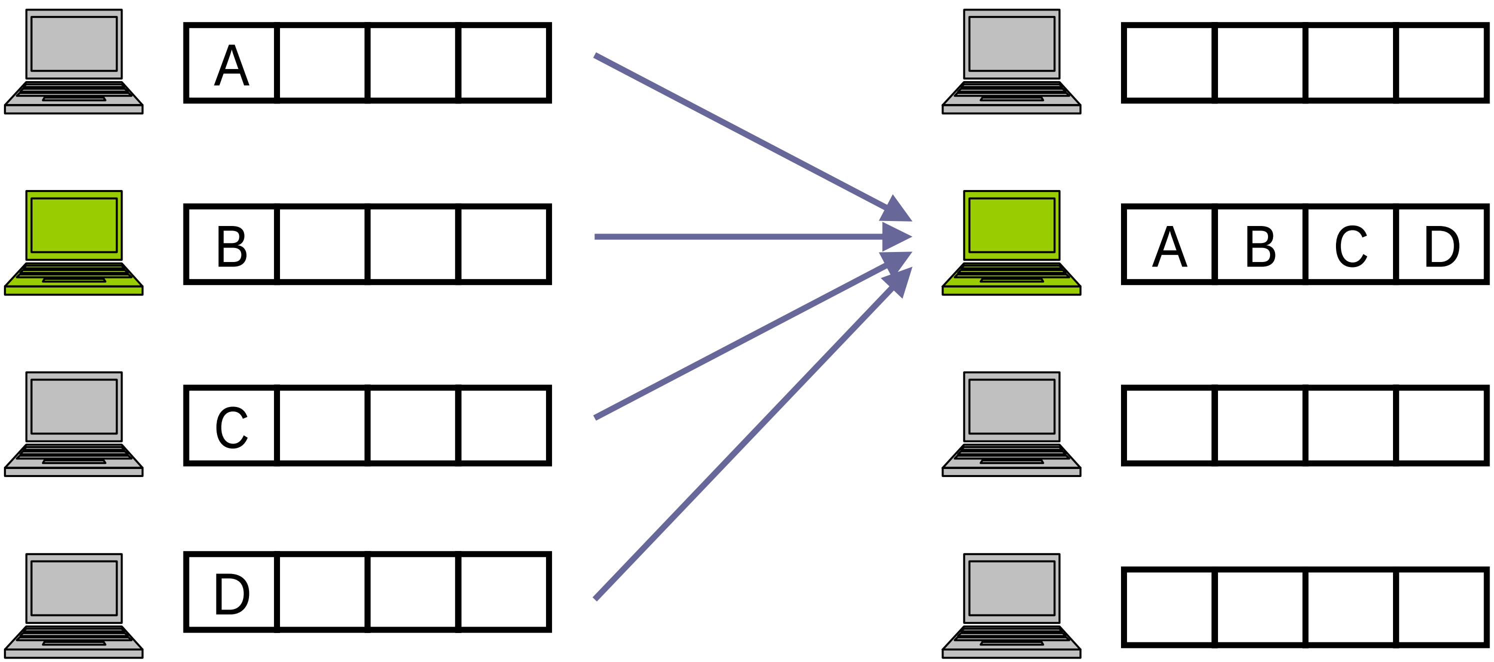
Collective Communication

- scatter
  - data from one process are distributed among all processes
  - example: rows of a matrix for a parallel solution



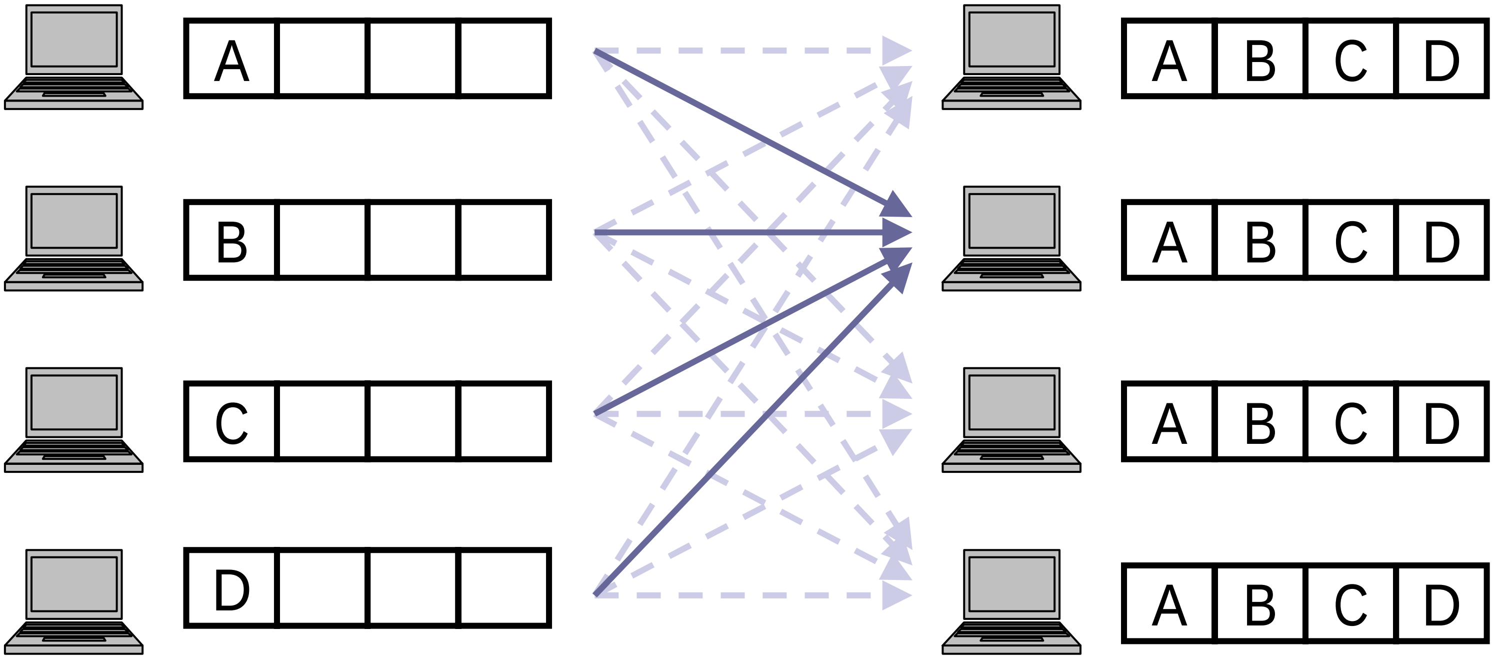
Collective Communication

- gather
  - data from all processes are collected by one process
  - example: assembly of solution vector from parted solutions



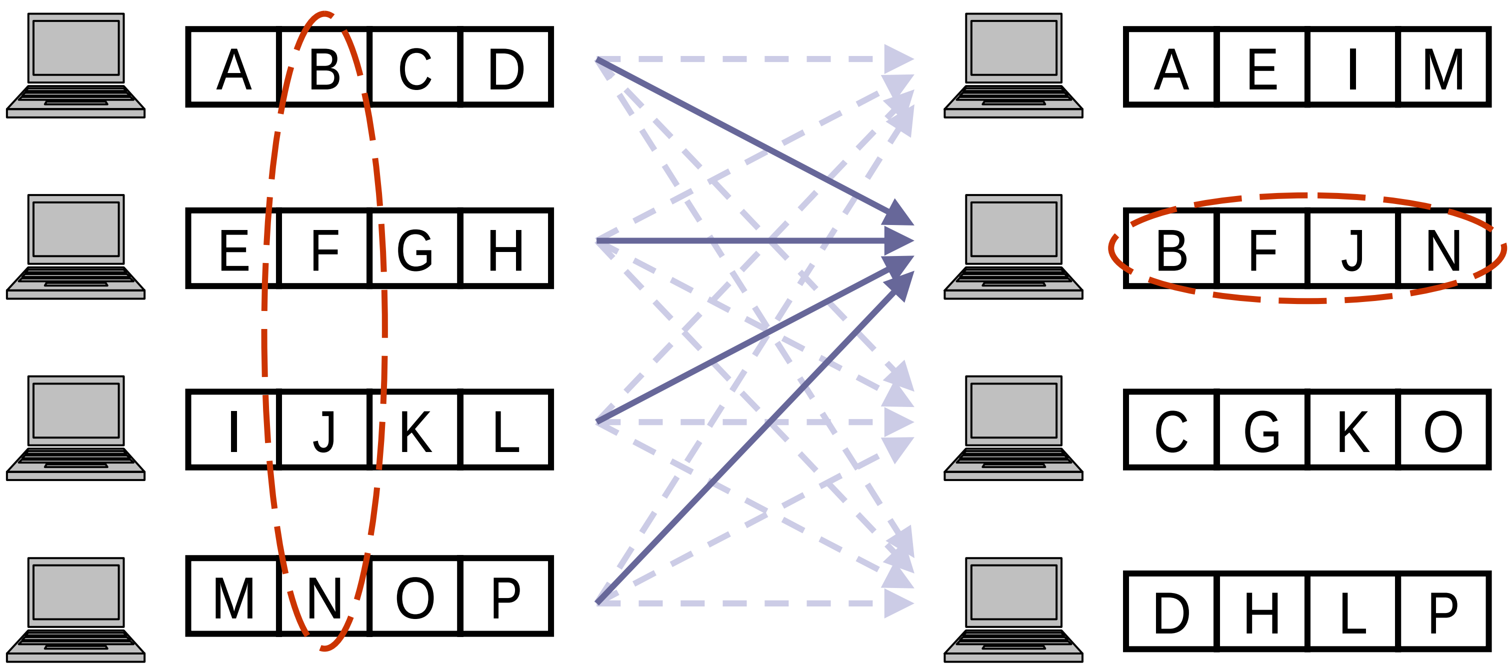
Collective Communication

- gather-to-all
  - all processes collect distributed data from all others
  - example: as bf., processes need solution for continuation



Collective Communication

- all-to-all
  - data from all processes are distributed among all others
  - example: any ideas?



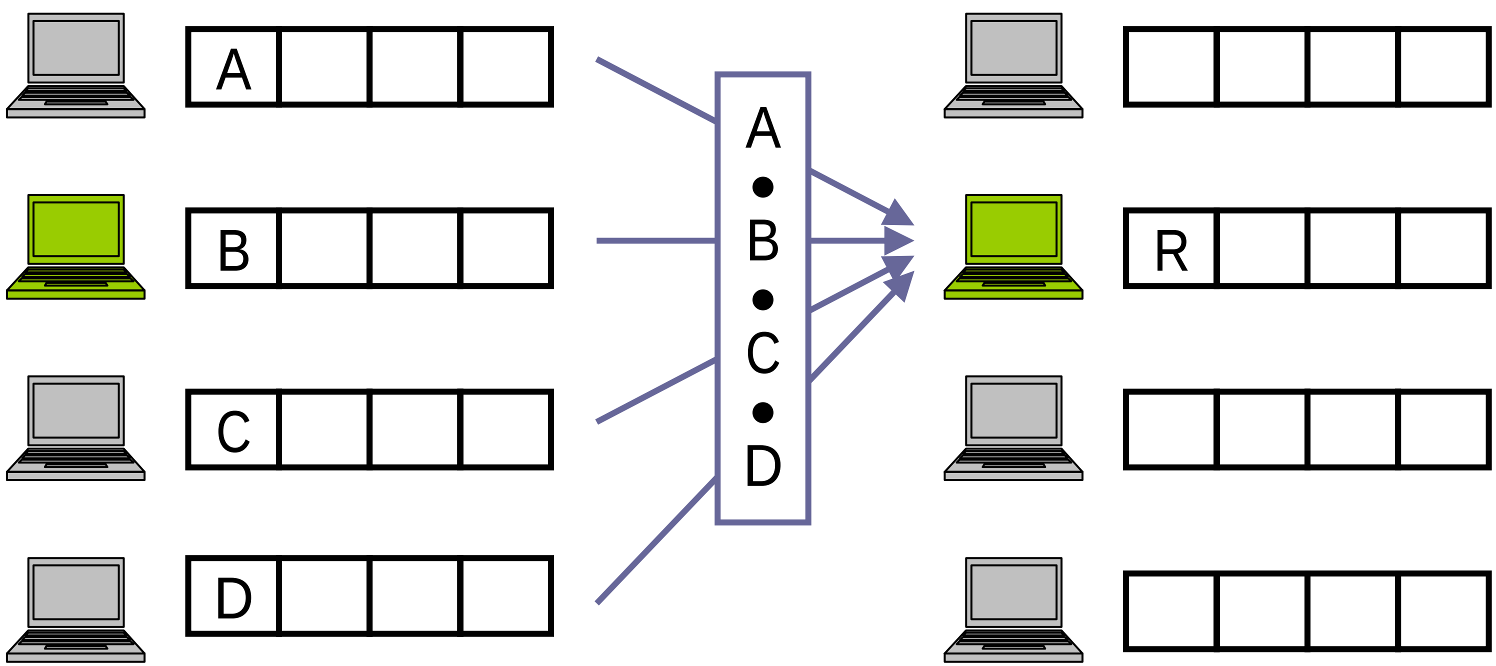
Collective Communication

- all-to-all (cont'd)
  - all-to-all also known as *total exchange*
  - example: transposition of matrix
    - matrix *A* stored row-wise in memory
    - processes transpose blocks in parallel
    - total exchange of blocks

$$A = \begin{pmatrix} a & b & c & d \\ e & f & g & h \\ i & k & l & m \\ n & o & p & q \end{pmatrix} \rightarrow A^T = \begin{pmatrix} a & e & i & n \\ b & f & k & p \\ c & g & l & p \\ d & h & m & q \end{pmatrix}$$

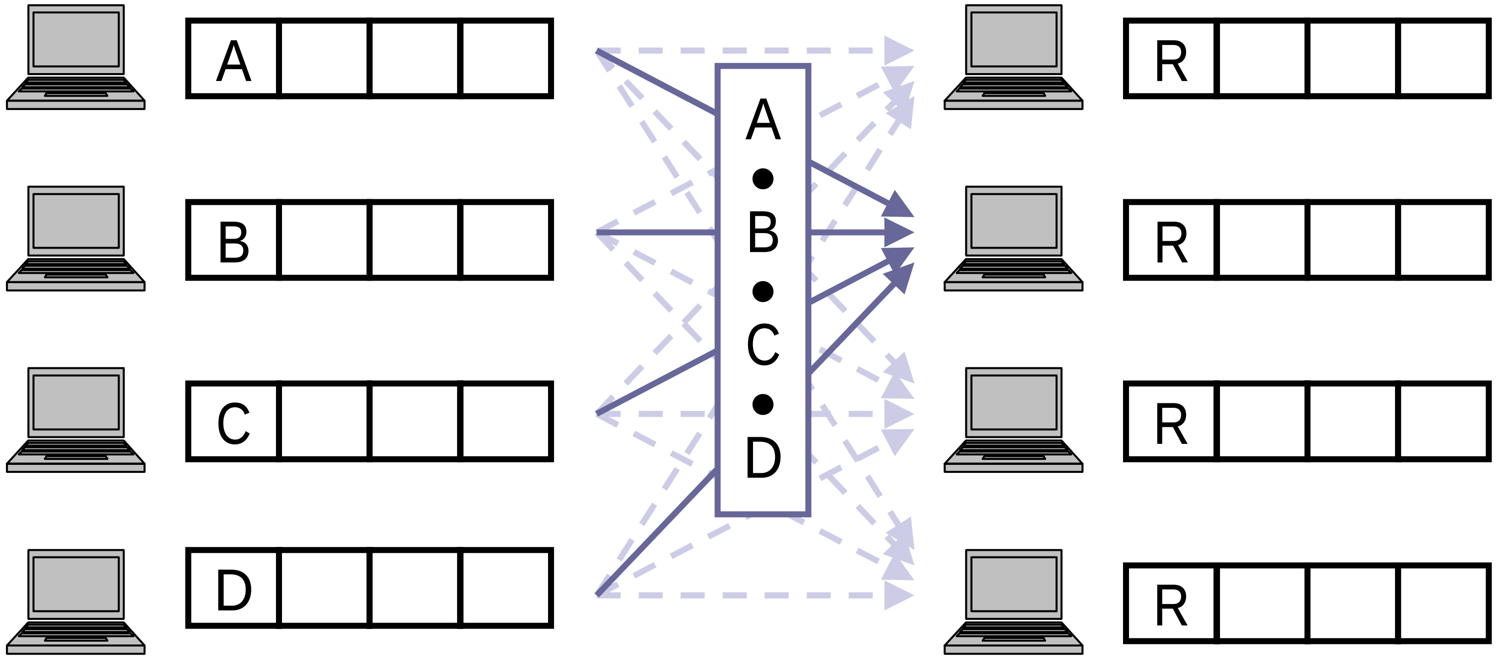
Collective Communication

- reduce
  - data from all processes are reduced to single data item(s)
  - example: global minimum/maximum/sum/product/…



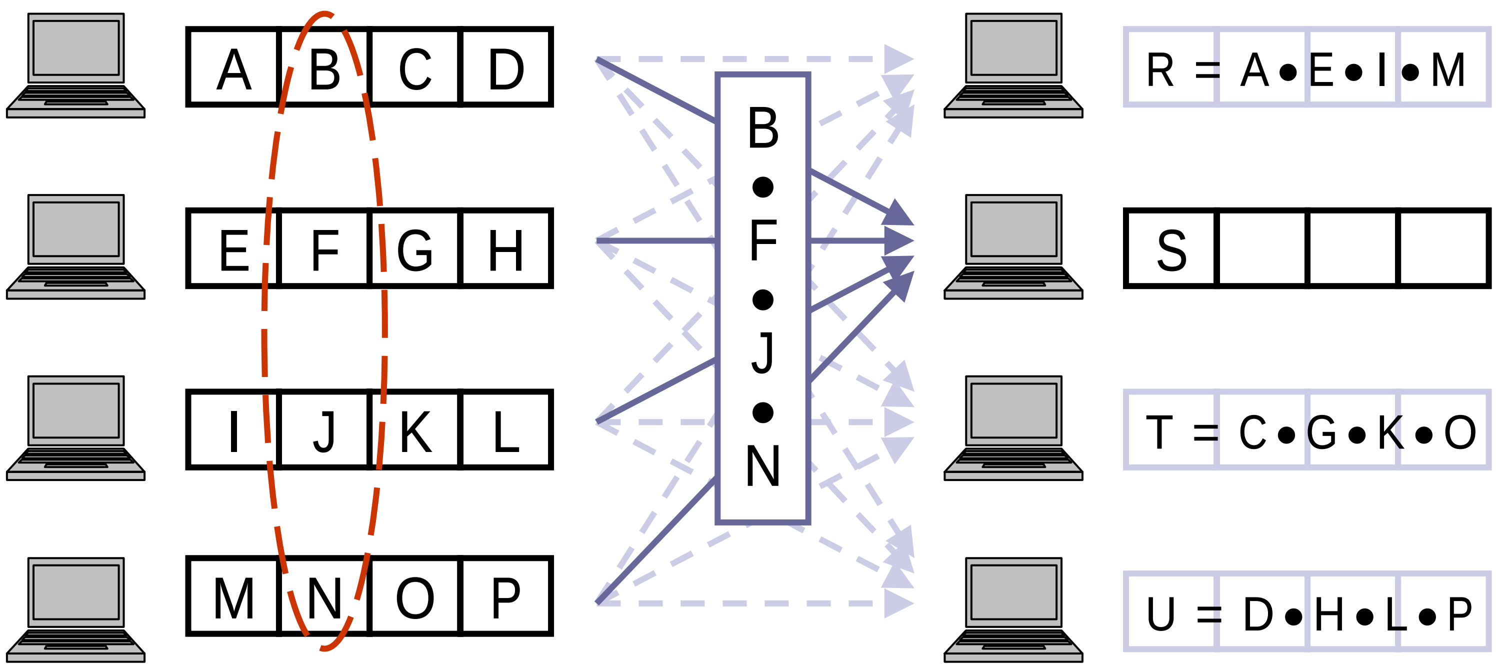
Collective Communication

- all-reduce
  - all processes are provided reduced data item(s)
  - example: processes need global minimum for continuation



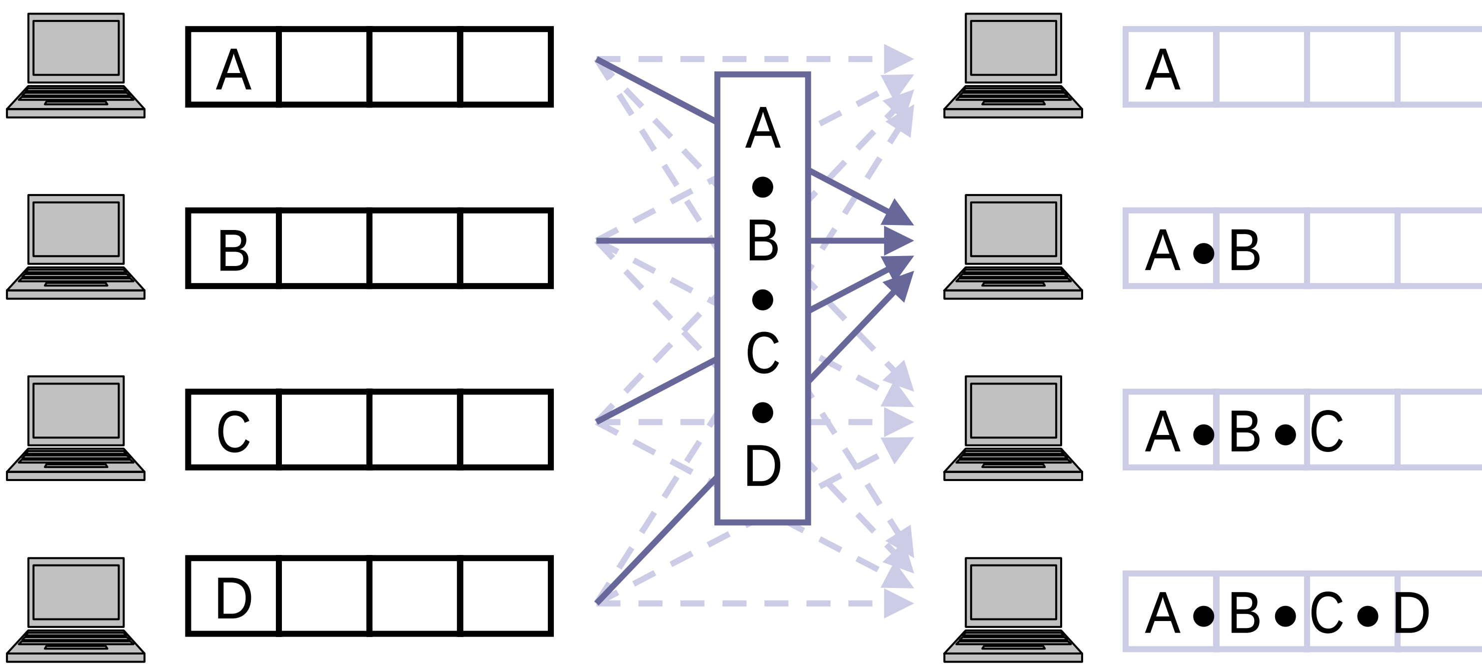
Collective Communication

- reduce-scatter
  - data from all processes are reduced and distributed
  - example: any ideas?



Collective Communication

- parallel prefix
  - processes receive partial result of reduce operation
  - example: carry look-ahead for adding two numbers



Collective Communication

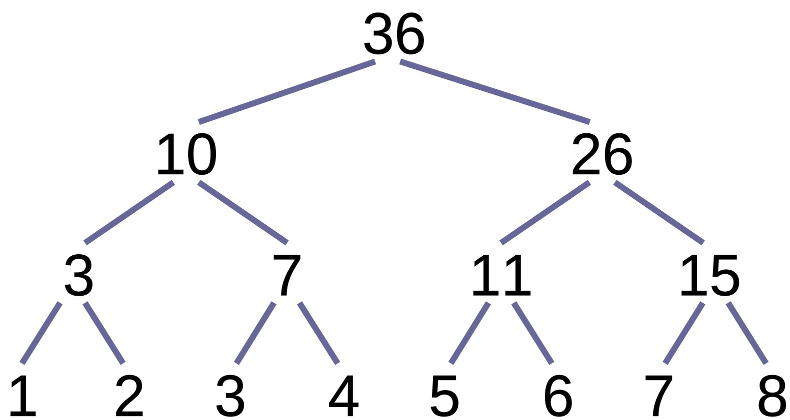
- parallel prefix (cont'd)
  - carrying problem (overflow) when adding two digits
  - consider

c_3	c_2	c_1	c_0		Carry
	a_3	a_2	a_1	a_0	First Integer
	b_3	b_2	b_1	b_0	Second Integer
s_4	s_3	s_2	s_1	s_0	Sum

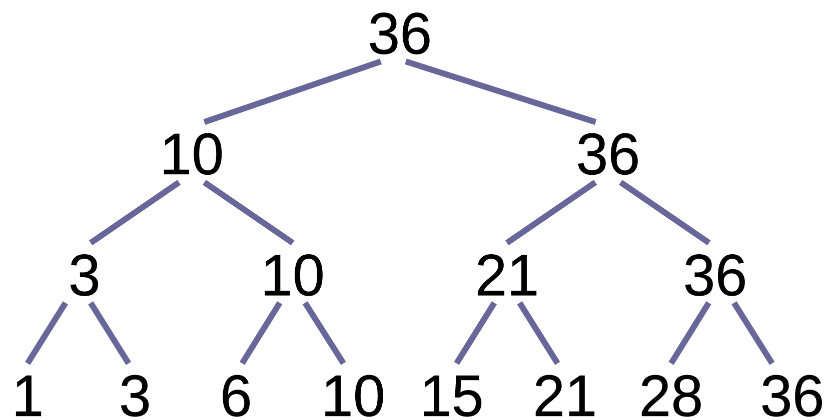
  - c_i can be computed from c_{i-1} by $(a_i + b_i) \cdot c_{i-1} + a_i \cdot b_i$
  - carry look-ahead: each c_i computed by parallel prefix
  - afterward, s_i are calculated in parallel

Collective Communication

- parallel prefix (cont'd)
  - implementation with binary trees
  - example: finding all (partial) sums in O(log *n*) time



up the tree



down the tree (prefix)

→ even entries: $b_i = c_i$

→ odd entries: $b_i = c_{i-1} + b_i$

Parallel Sessions

- lectures
  - Advanced MPI Programming (R.-P. Mundani)
  - MPI Tools (I. L. Muntean)
  - Advances in Cluster Computing (I. L. Muntean)
  - Tuning Parallel Algorithms (I. L. Muntean)
  - Computational Steering (R.-P. Mundani)
  - Studying in Germany (R.-P. Mundani)
- exercises
  - MPI Exercises Part I & II (2x) (Mundani/Muntean)
  - Advanced MPI Exercises (R.-P. Mundani)
  - Numerics Exercises (I. L. Muntean)
  - Introduction to Linux Part I & II (2x) (A. Mors)
  - Linux Server Administration (A. Mors)
  - SimLab Administration (Closed Session) (A. Mors)

mundani@in.tum.de

http://www5.in.tum.de/~mundani/