Concurrent Computing

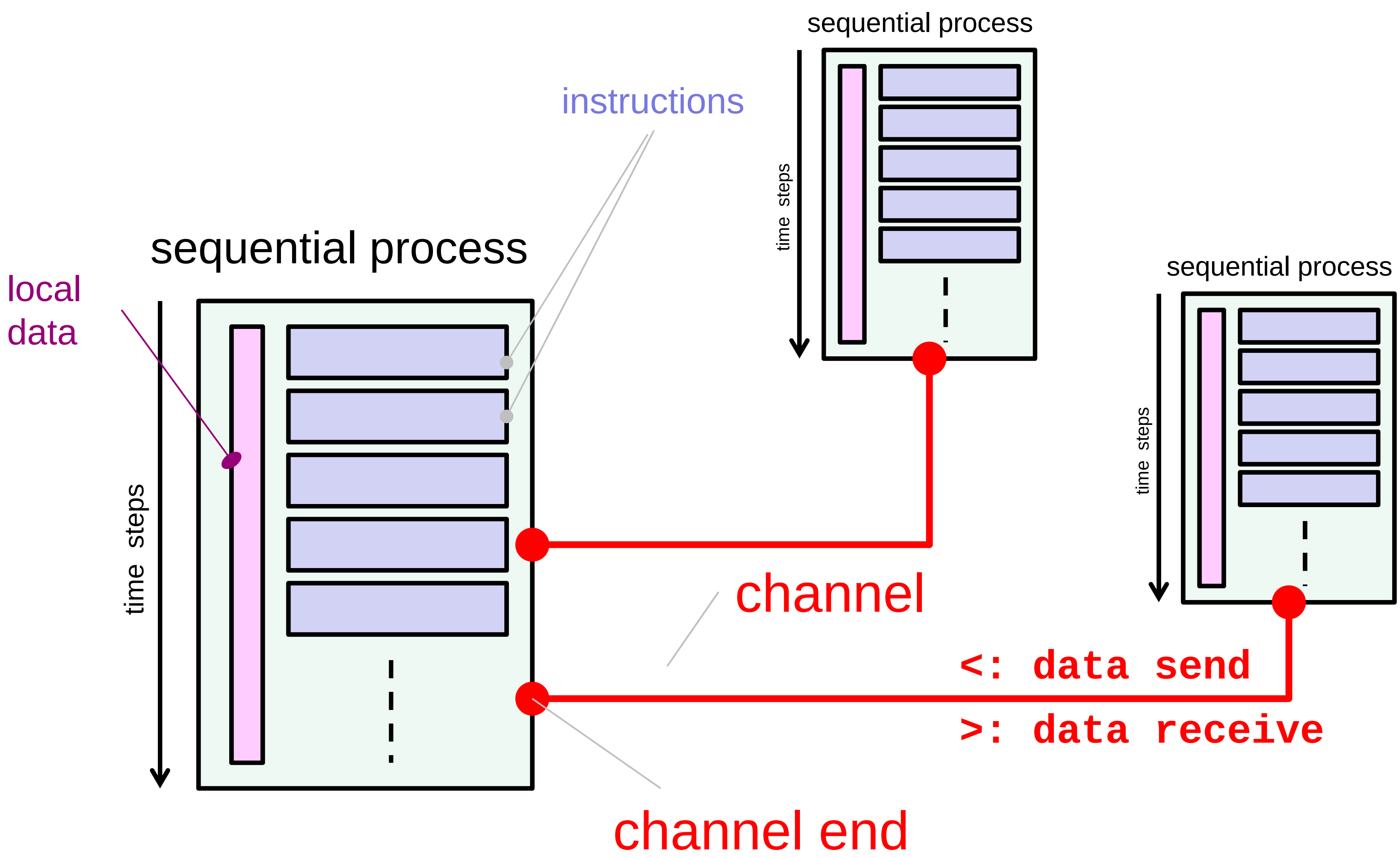
Lecturers: Prof. Majid Mirmehdi majid@cs.bris.ac.uk)
Dr. Tilo Burghardt tilo@cs.bris.ac.uk
Dr. Daniel Page page@cs.bris.ac.uk

Web: http://www.cs.bris.ac.uk/Teaching/Resources/COMS20001

LECTURE MM4

BASICS OF CONCURRENT SYSTEM DESIGN

The Building Blocks: Processes and Channels

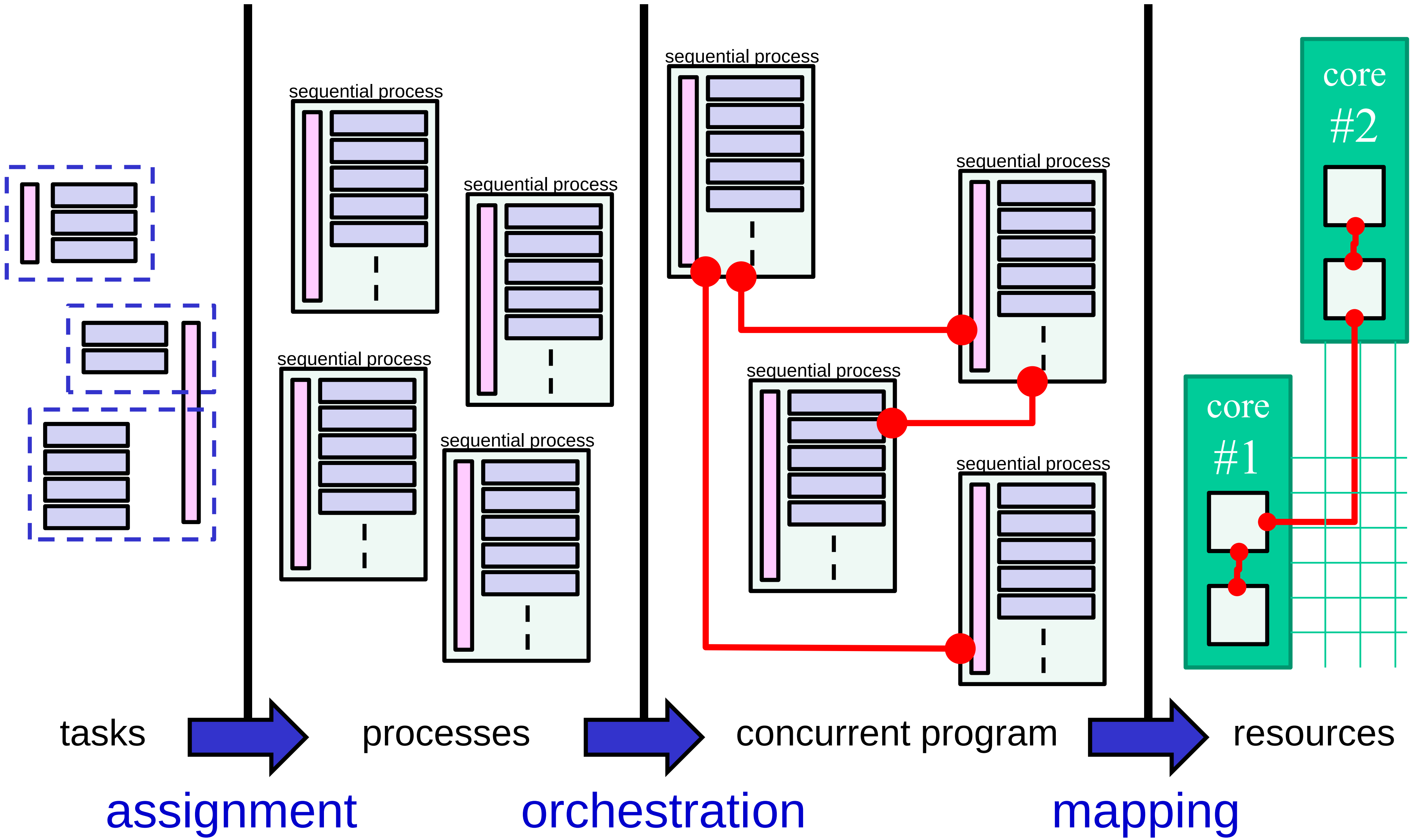


Stages in Designing a Concurrent System

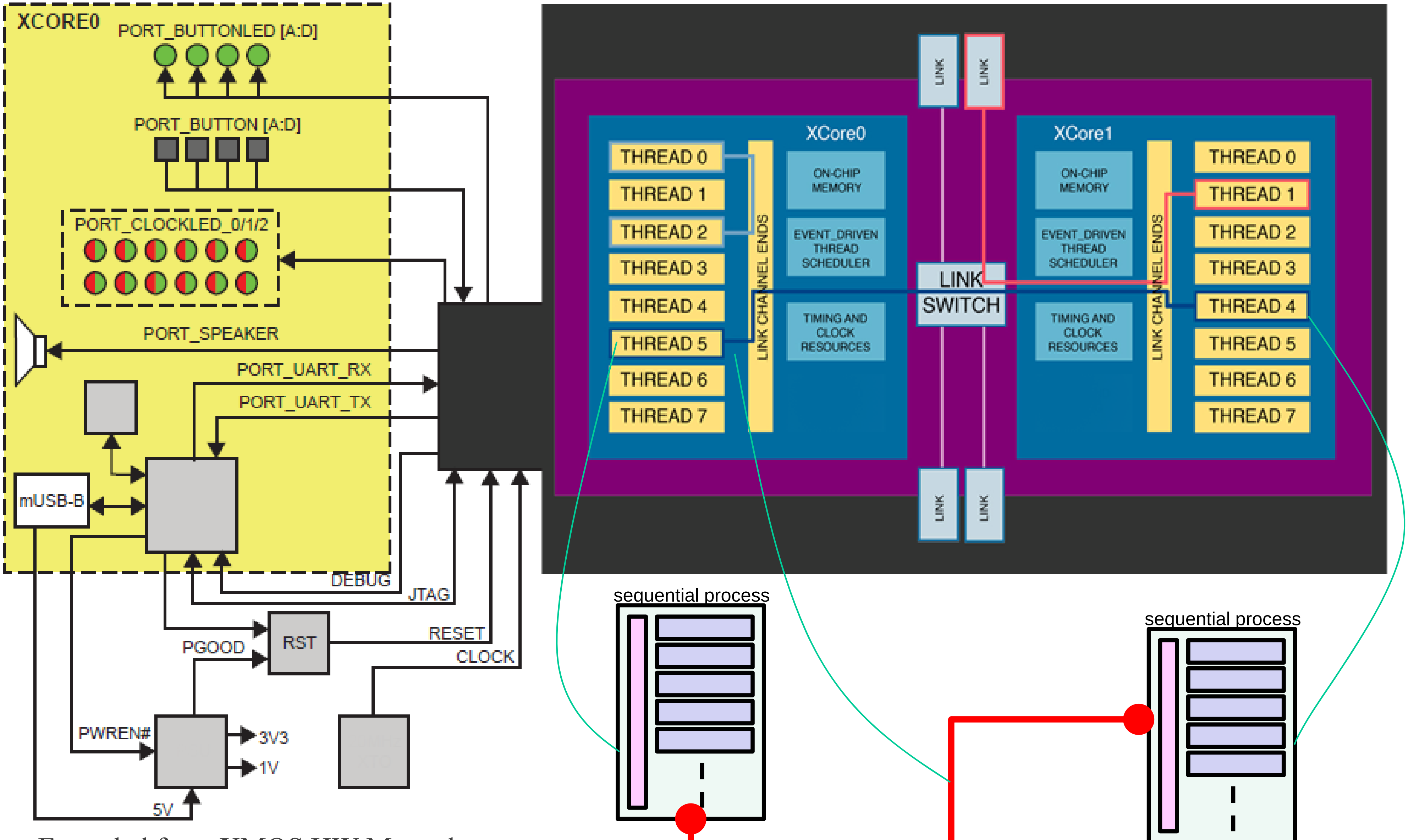
Three fundamental stages:

- Assignment of application tasks to processes
- Orchestration of processes into a concurrent program
- Mapping of program onto available resources

Stages in Designing a Concurrent System



Mapping to Hardware on XMOS XS-1



Extended from XMOS HW Manual

Basic Example: Distance Evaluation

- Assume that the task is to design a concurrent system that uses two processes efficiently in order to calculate distances between:
  - ▶ **patrol locations** stored in an array **A**
  - ▶ **emergency locations** stored in an array **B**

  and finally outputs the 2D result array **C**

```
// SEQUENTIAL SOLUTION (FRAGMENT)
...
typedef point...;       //define point structure
point   A[4]={...};     //initial locations A
point   B[4]={...};     //initial locations B
int     C[4][4]={...}; //initial result matrix C

// calculate all distances
for (int i = 0; i<4; i++)
  for (int j = 0; j<4; j++)
    C[i][j] = dist(A[i], B[j]);

// output result C
...
```

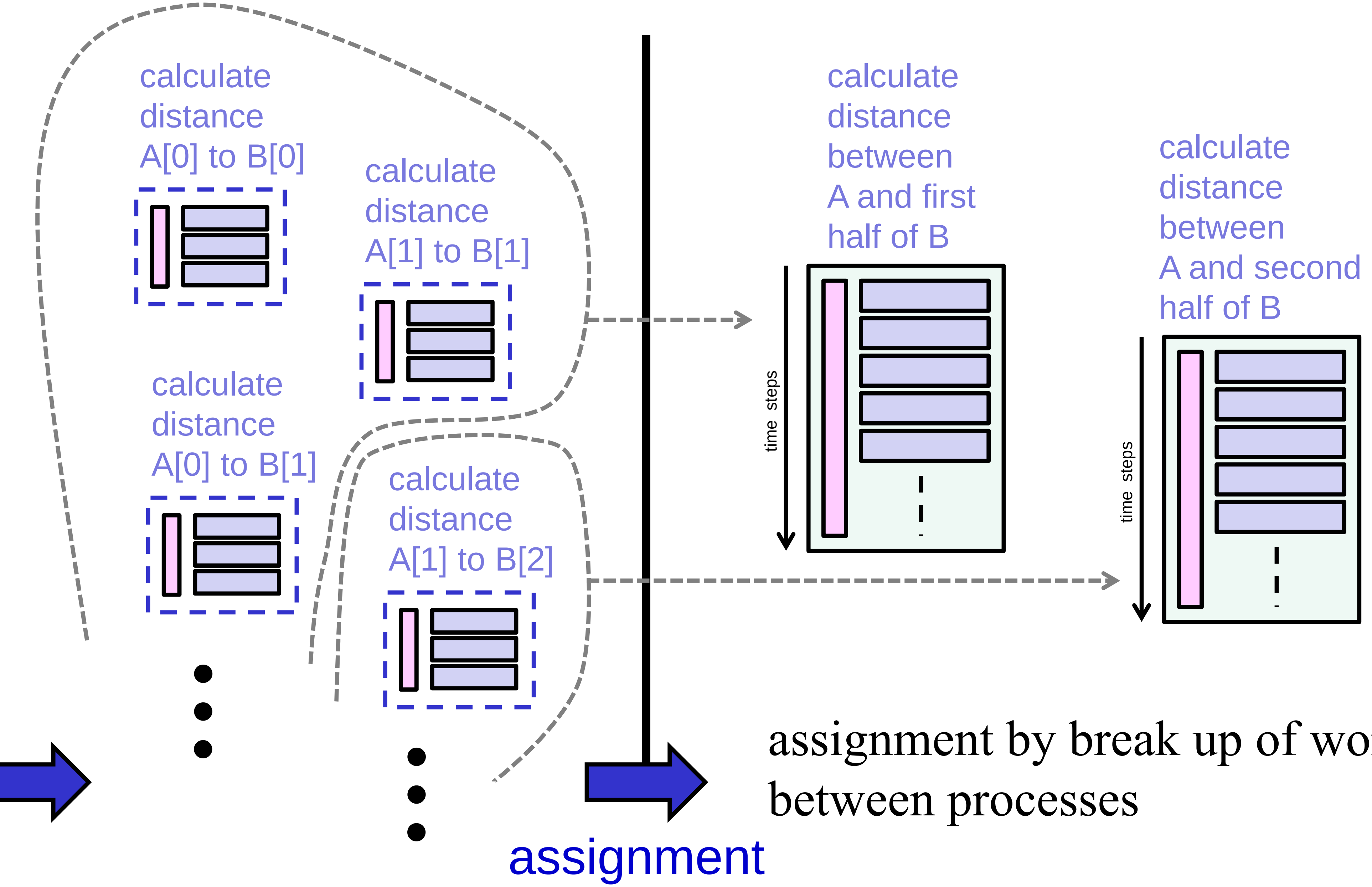
Problem Decomposition & Task Assignment

- **Creation of Process Candidates**
  (i.e. identifying tasks and partitioning the task space into processes by bundling tasks)

→ **Criteria:** minimum coupling, maximum compactness
  - Goal: clustering tasks of similar functionality demanding similar data portions
  - Analogy: design of objects in OO programming

→ **Target:** balance work and reduce communication during runtime
  - application often hints at good process candidates

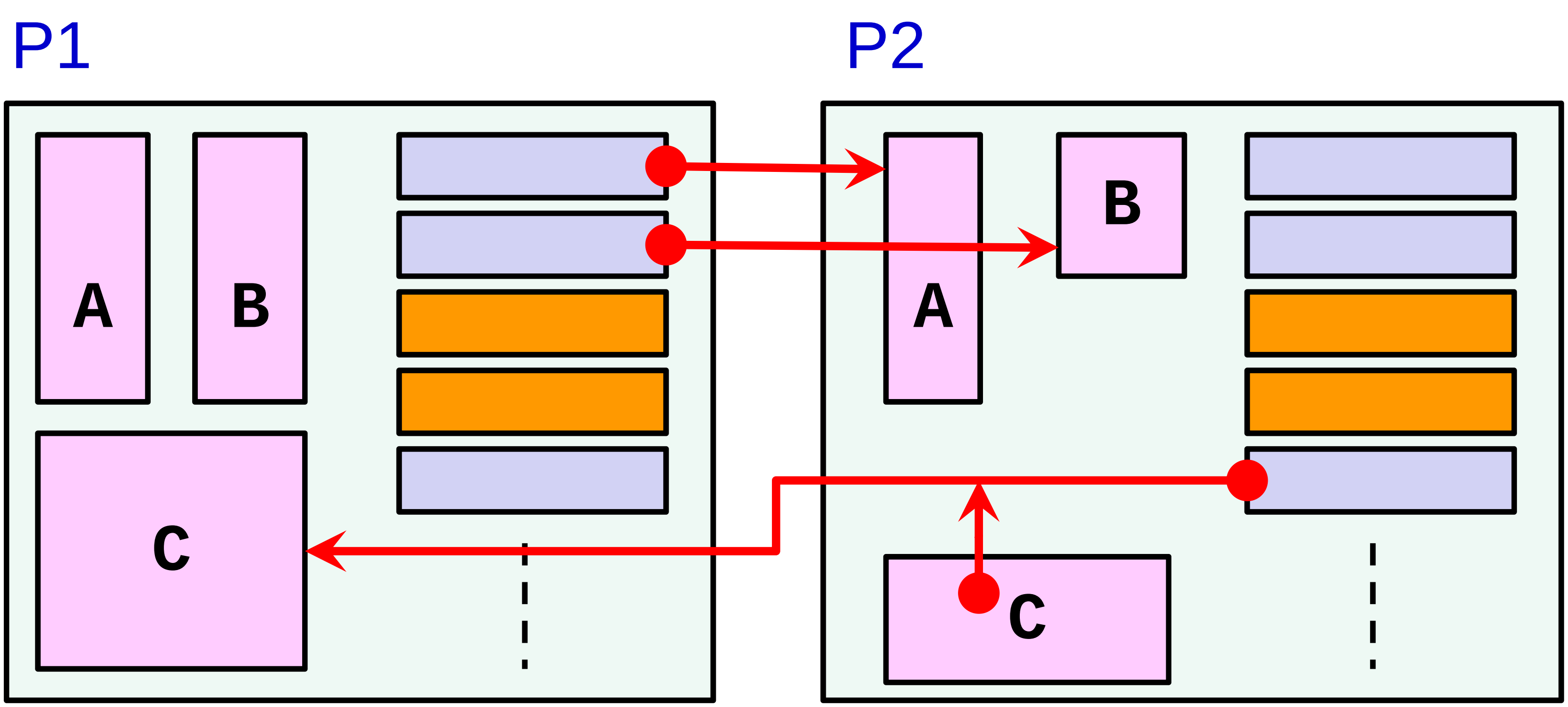
Example Task Assignment



assignment by break up of work between processes

Basic Example: Process Orchestration

- **Orchestration:** determine communication strategy and channels
  - e.g. P1 sends data to P2
  - P1 and P2 compute
  - P2 sends output to P1



Concurrent *XC* Fragment showing Data Flow

```
// process P1 (fragment)
void processOne(chanend ch) {
  point A[4]={...}; //init A
  point B[4]={...}; //init B
  int   C[4][4]={...}; //init C

  // send data to process two
  for (int i = 0; i<4; i++)
    ch <: A[i];
  for (int i = 0; i<2; i++)
    ch <: B[i];

  // calculate some distances
  for (int i = 0; i<4; i++)
    for (int j = 2; j<4; j++)
      C[i][j] = dist(A[i], B[j]);

  // receive results
  for (int i = 0; i<4; i++)
    for (int j = 0; j<2; j++)
      ch :> C[i][j];

  ... //output result C
}
```

```
// process P2 (fragment)
void processTwo(chanend ch) {
  point A[4]; //allocate local memory
  point B[2];
  int   C[4][2]={...};

  // receive data from process one
  for (int i = 0; i<4; i++)
    ch :> A[i];
  for (int i = 0; i<2; i++)
    ch :> B[i];

  // calculate some distances
  for (int i = 0; i<4; i++)
    for (int j = 0; j<2; j++)
      C[i][j] = dist(A[i], B[j]);

// send results
  for (int i = 0; i<4; i++)
    for (int j = 0; j<2; j++)
      ch <: C[i][j];
}
```

channel ch

Trade-off: Parallelisation vs. Communication

- in the example, distance calculations between points are independent of each other (…and thus, can be parallelised easily…'embarrassingly' parallel)

- Dividing work over **N** processes does not lead to **N**-times speedup as there are communication demands:
  - e.g. copy of **A** needs to be sent to all processors + copy of subset of **B** to each processor + results need collecting

→ COMMUNICATION is NOT free…

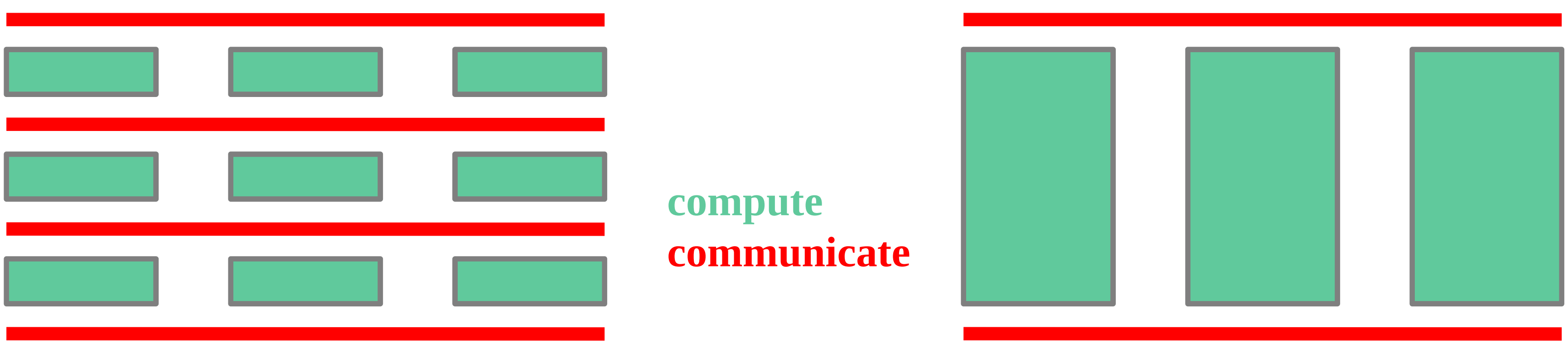
…it creates overheads and takes time!

Impact of Granularity: Fine vs. Coarse

- computation is typically separated by periods of communication

- **Fine-grain System**
  - Small computation to communication ratio
  - Little computational work between communication stages
  - Less opportunity for computation enhancement
  - High communication overhead
  - Good for reactive & dynamic problems

- **Coarse-grain System**
  - Large computation to communication ratio
  - Big blocks of computational work between communications
  - More opportunity for performance increase due to large blocks
  - Hard to load balance efficiently



Dependencies: Limits to Performance Scalability

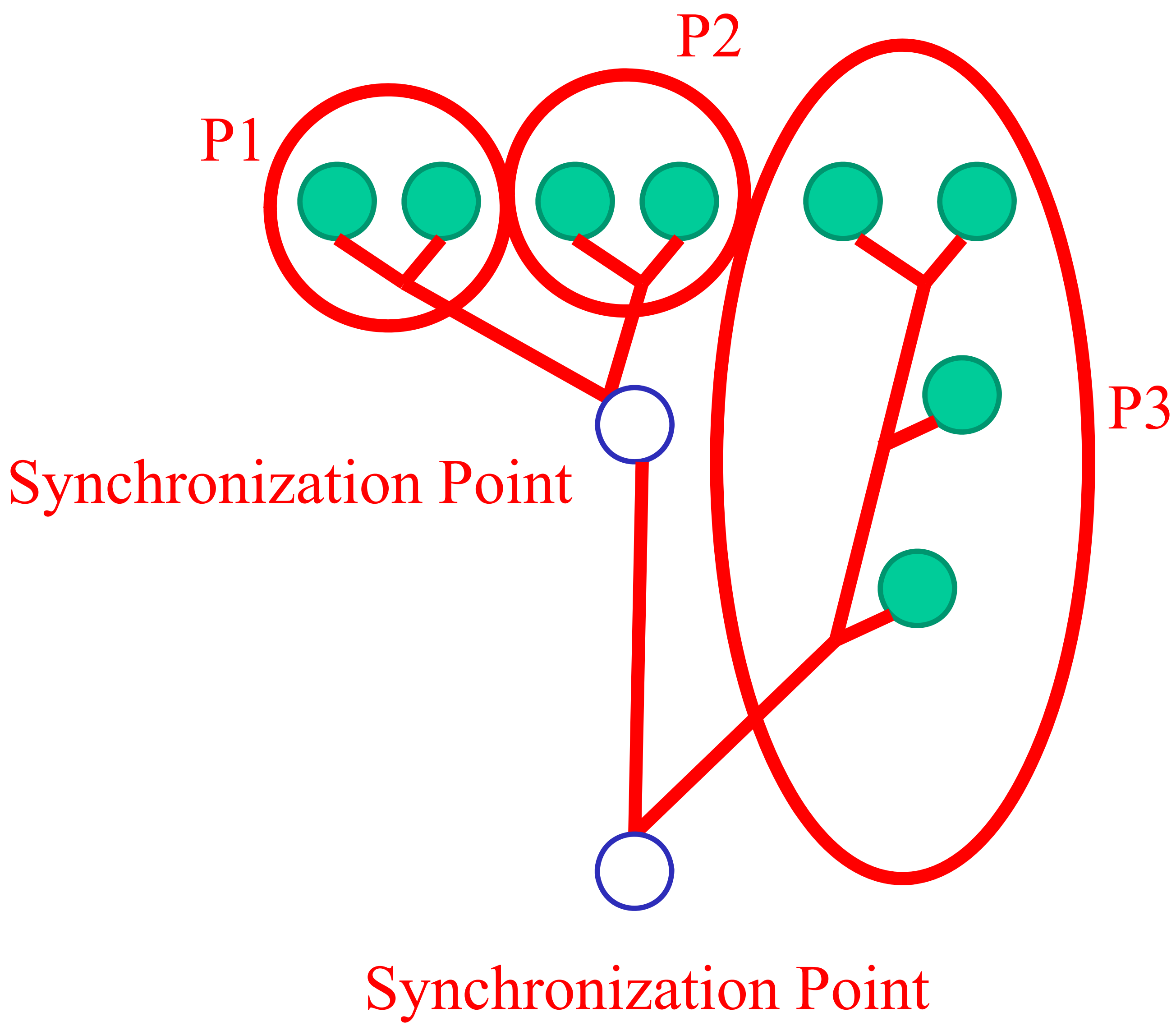
- So far, we have focussed on problems that are easily divided into parallel threads … an important design issue is **Data Dependency!**
- Example:

```
A = B + C
D = A + A
E = D / C
```

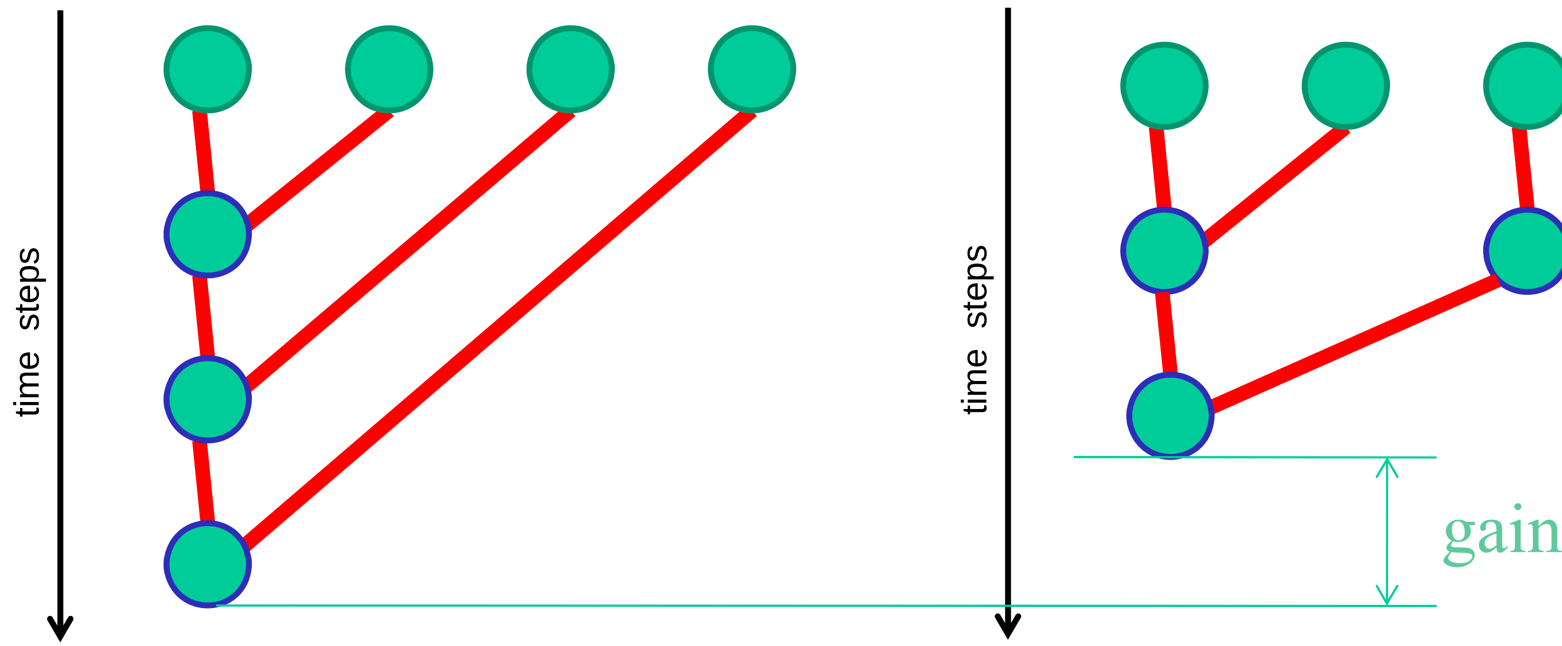
`One computation can not proceed until another is complete.`

- Dependencies between (interim) results enforce waiting at synchronisation points…
- Ordering and balancing of calculations can help minimise idle waiting for results…

Analysing Data Dependency: Reduction Graphs



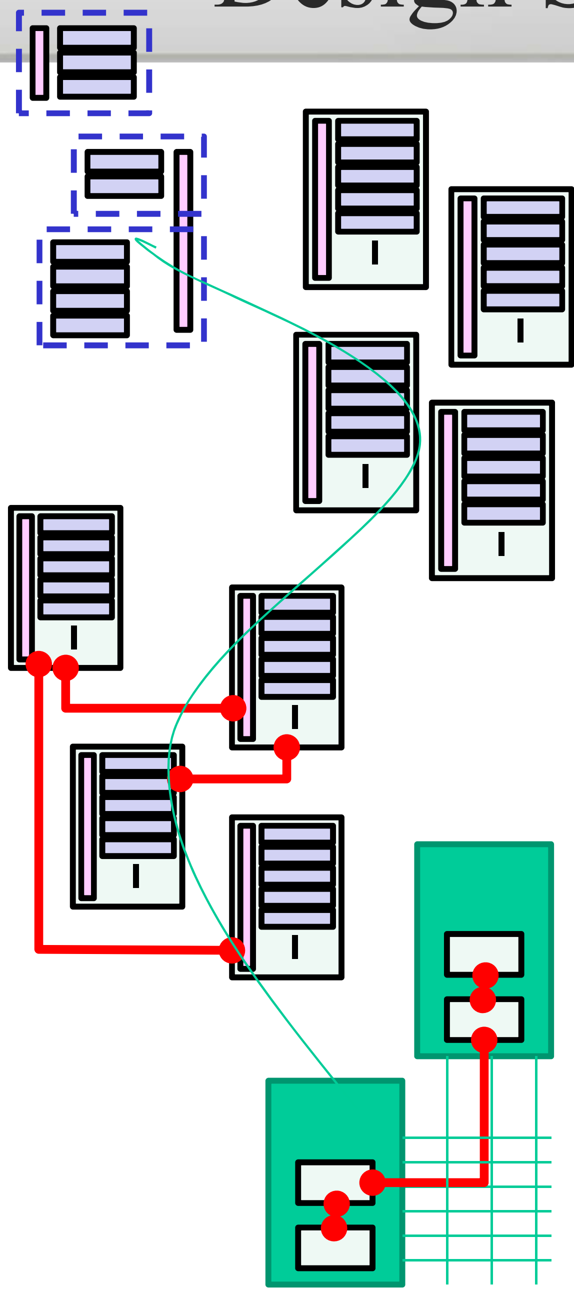
Serial Reduction vs. Tree Reduction

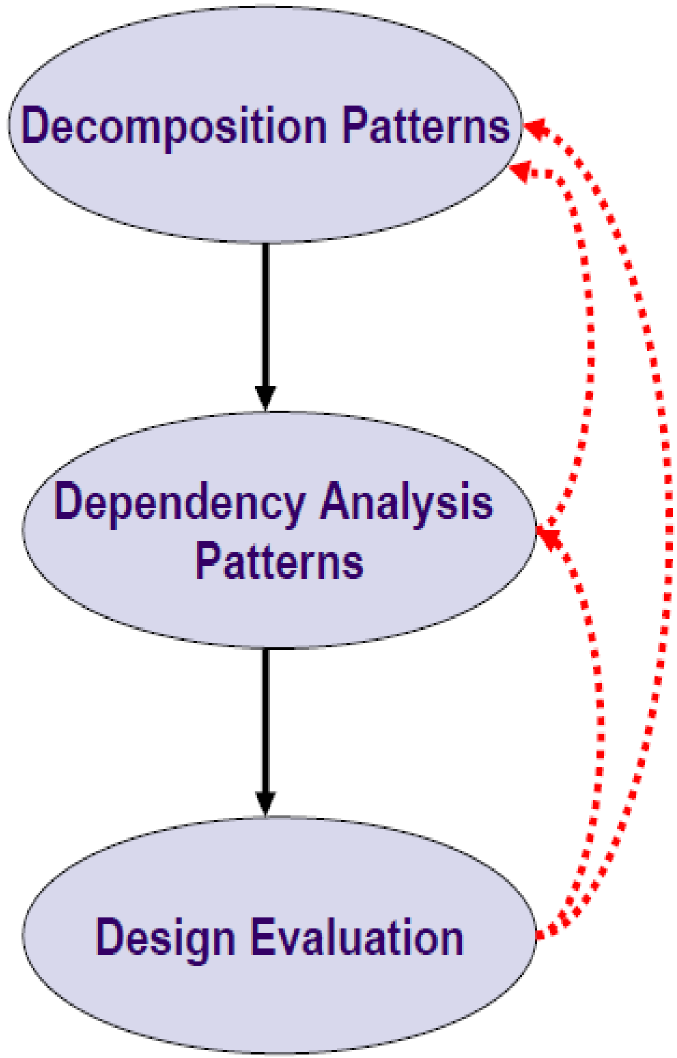


- Good for non-associative combination of interim values
- Usually followed by broadcast of result

- Good for associative combination of interim values
- N steps for 2^N elements
- Especially efficient when only one task needs result

Design Space for Concurrent Systems





- Processes formed are at right ***granularity***?
- Processes are ***compact*** operating on local data?
- Communication ***dependencies*** are well ***reduced***?
- Expected message ***traffic is low***?
- Hardware ***supports*** approach well?

Bernstein's Criteria for Task Independency

(What can be parallelised?)

A J Bernstein

R_i → set of memory locations read (input) by task T_i

W_j → set of memory locations written to (output) by task T_j

Two tasks T_1 **and** T_2 **are independent if:**

- input to T_1 is not part of output from T_2
- input to T_2 is not part of output from T_1
- outputs from T_1 and T_2 do not overlap

Example:

T_1

```
a = x + y
```

T_2

```
b = x + z
```

$R_1 = \{ x, y \}$
$W_1 = \{ a \}$

$R_2 = \{ x, z \}$
$W_2 = \{ b \}$

$$R_1 \cap W_2 = \phi$$

$$R_2 \cap W_1 = \phi$$

$$W_1 \cap W_2 = \phi$$