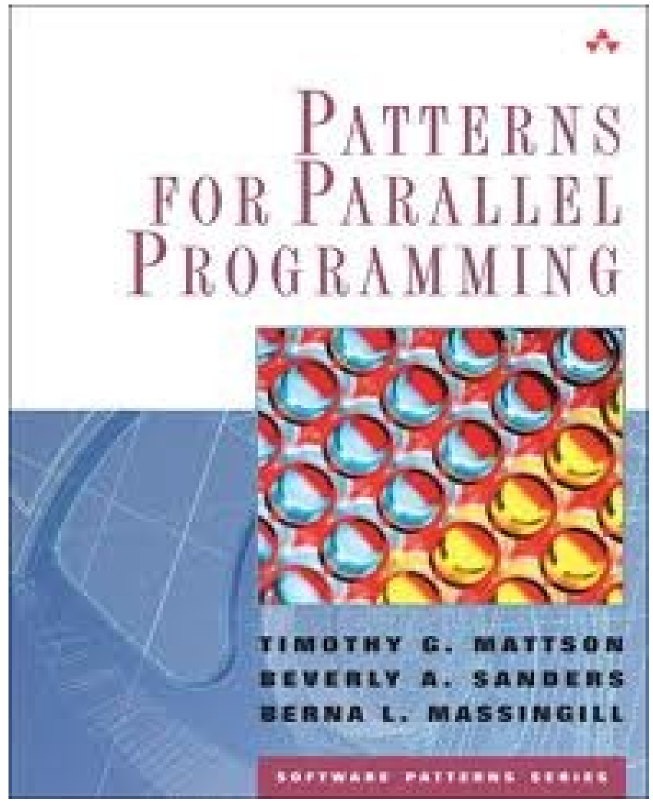
Patterns for Parallel Programming


Tim Mattson
Intel


Beverly A. Sanders
University of Florida

Patterns for Parallel Programming

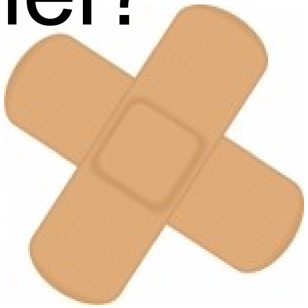


Timothy Mattson , Beverly Sanders , Berna Massingill,
Patterns for parallel programming, Addison-Wesley Professional, 2004
ISBN-13: 978-0321228116

University of BRISTOL

Sticking Plaster Pitfall

"Could you just tweak my serial code to make it run in parallel?"



University of BRISTOL

Why Bother With This Book?

- Recipe based
  - Recipes guide our thinking
  - Help us not to forget
- Introduces recurrent themes and terminology
  - e.g. (memory) latency, “loop parallelism”
- Emphasises ***design***
  - Amdahl's law highlights the pitfalls of looking for sticking-plaster speed-ups in serial programs – design for concurrency

Familiar Mantras - ..only more so

Flexibility

Environments will be more heterogeneous.

Efficiency

We're going parallel for a speed-up, right?

But more pitfalls (latency, thread overheads etc.)

Simplicity

Parallel codes will be more complicated.

All the more reason to strive for maintainable, understandable programs.

Four Design Spaces

Finding Concurrency

↕

Algorithm Structure

↕

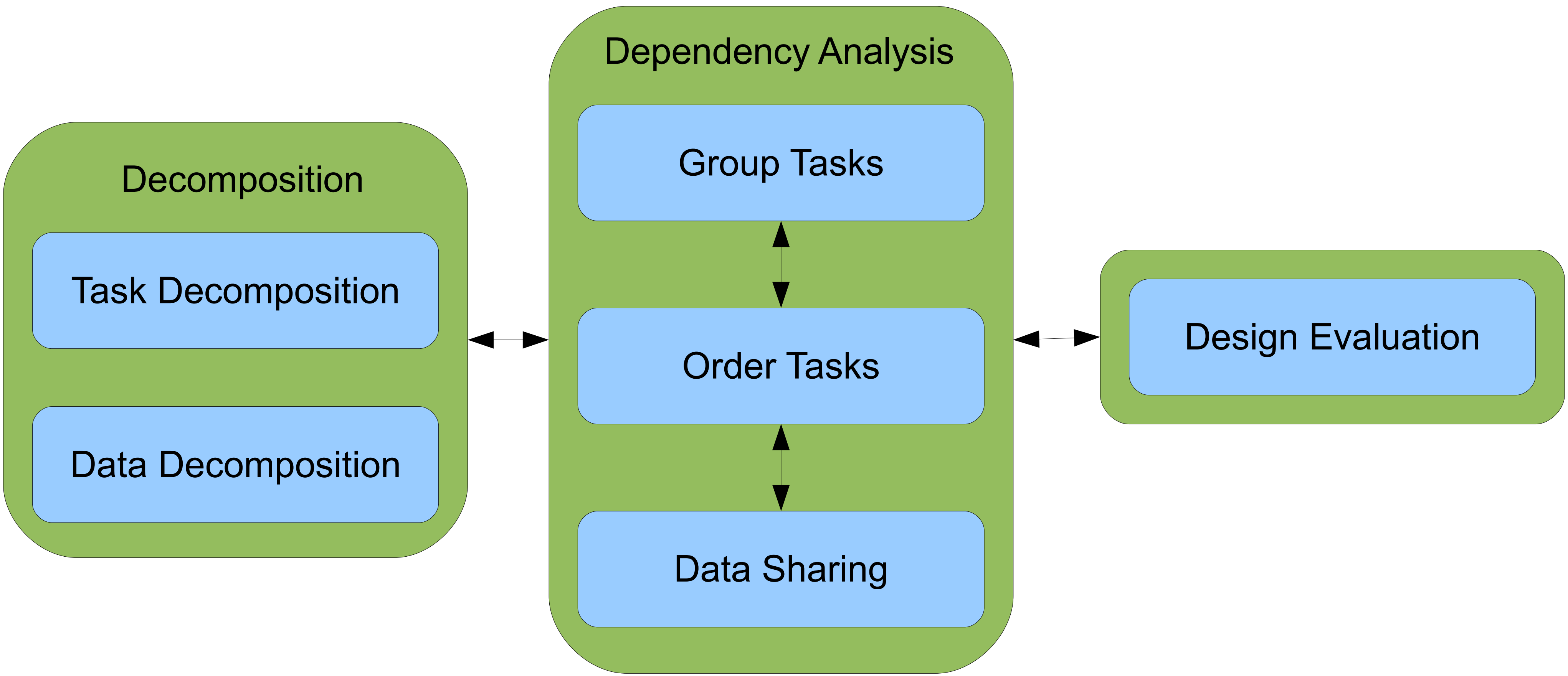
Supporting Structures

↕

Implementation Mechanisms

University of BRISTOL

Finding Concurrency



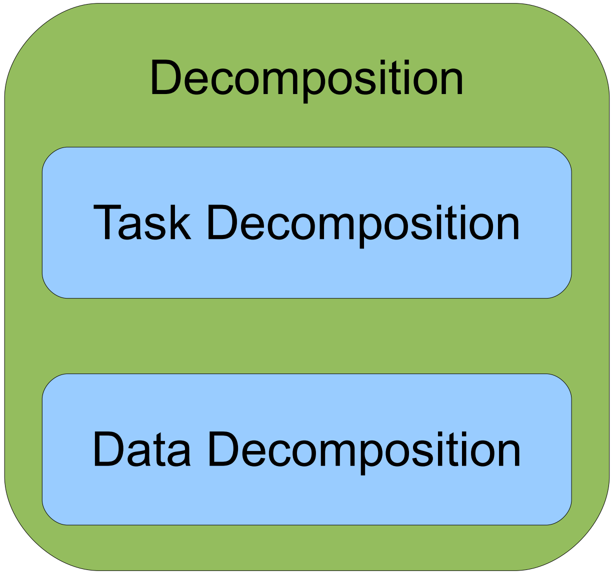
University of BRISTOL

Examples

- HPC: A Climate Model
- Embedded Systems: A Speech Recogniser
- The Cloud: Document Search

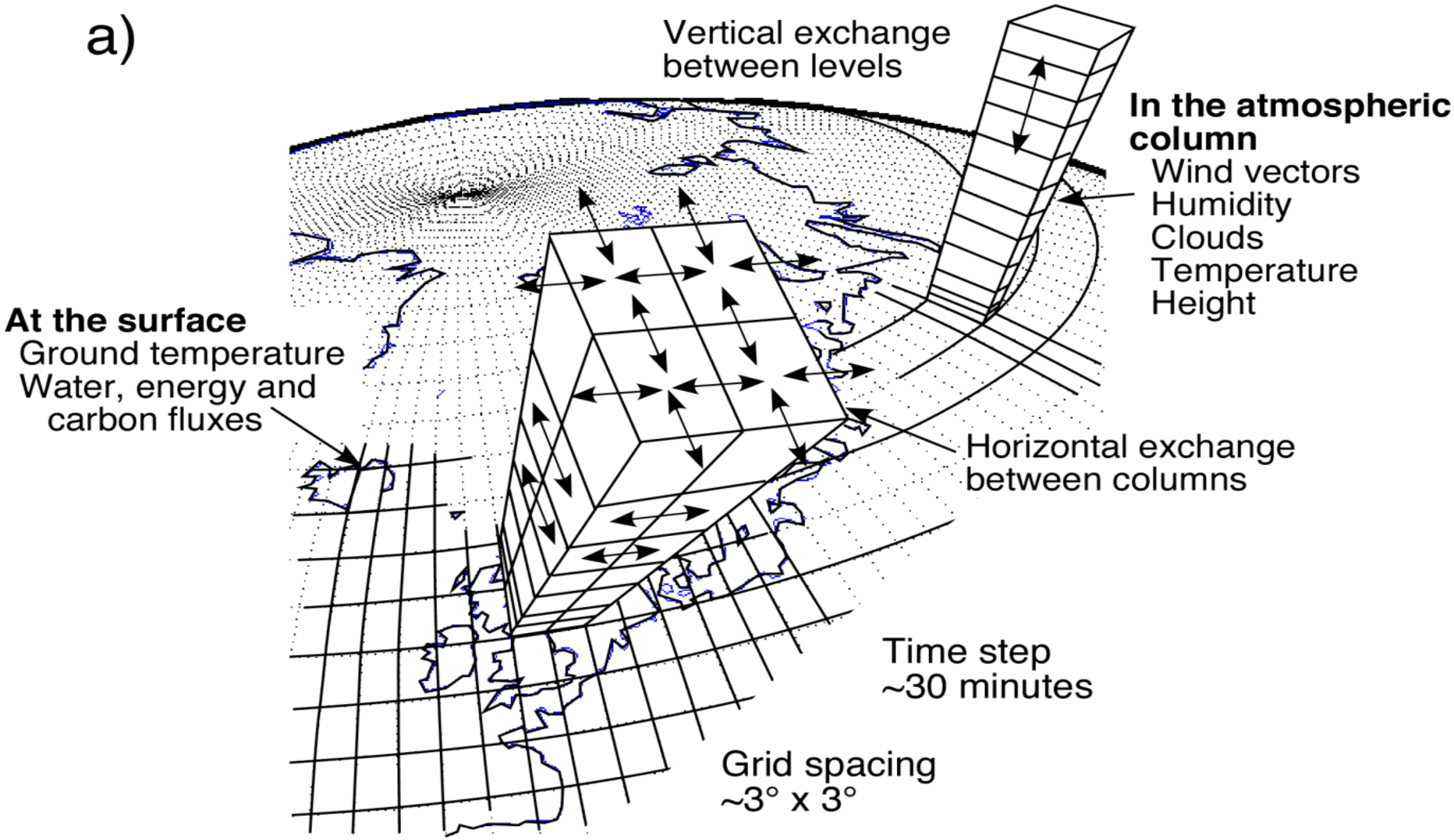
Highlights the fact that parallel programming is emerging everywhere..

Task vs. Data Decomposition



University of BRISTOL

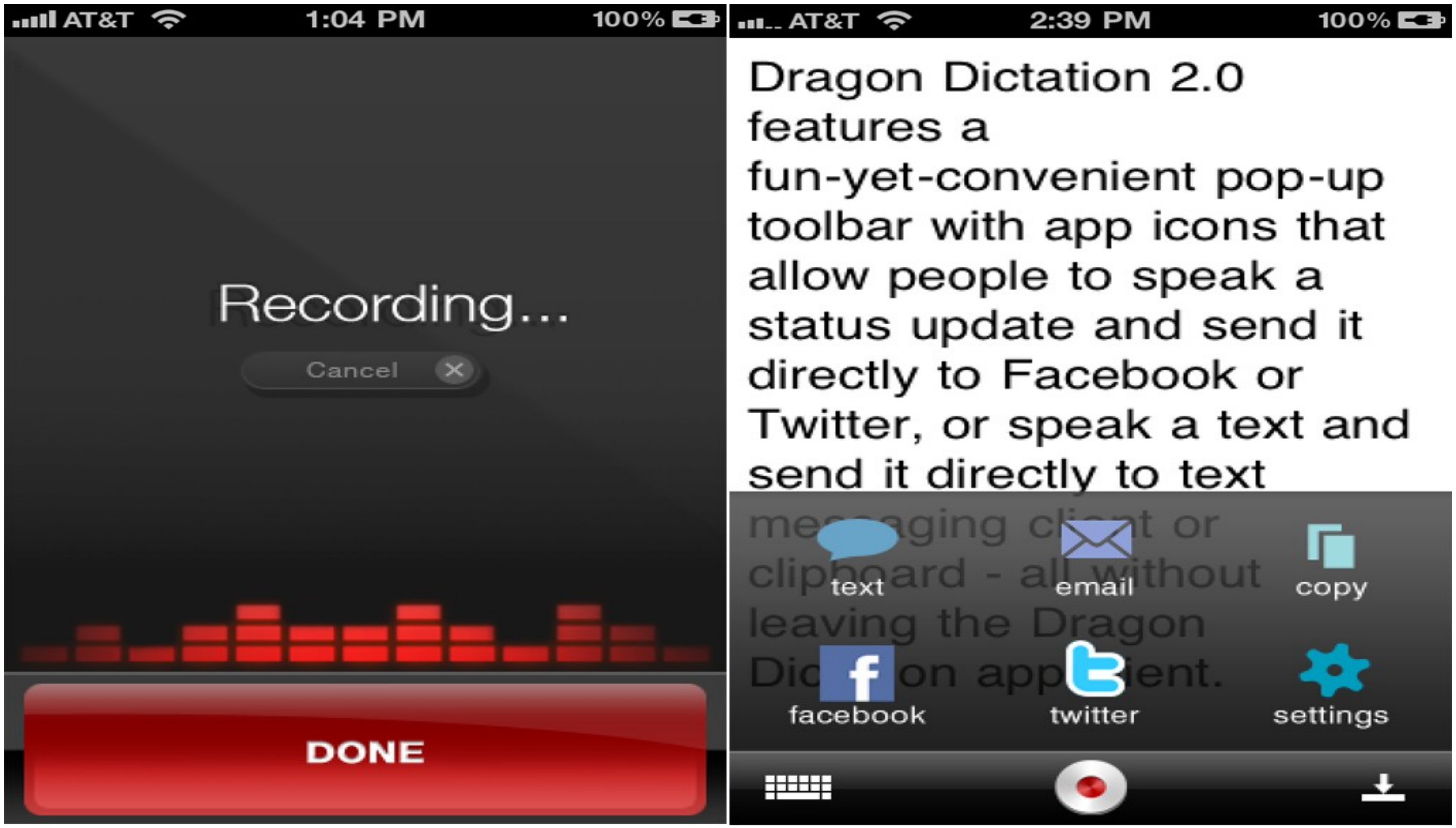
Data Decomposition (trad. HPC): A Climate Model



Data Parallel over grid cells

University of BRISTOL

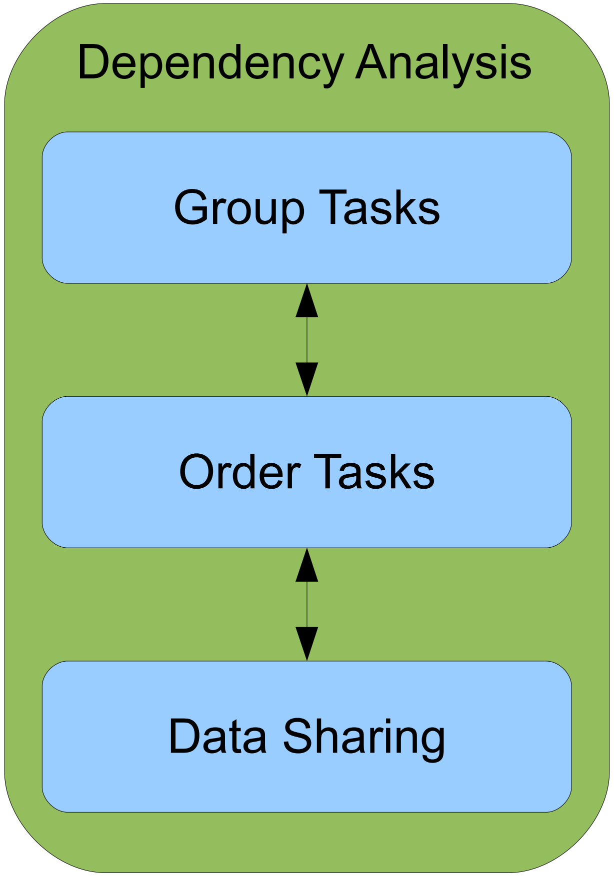
Task Decomposition (Embedded): A Speech Recogniser



Acoustic Analysis: concurrency in stages and components
Pattern Matching: search over many possible word matches

University of BRISTOL

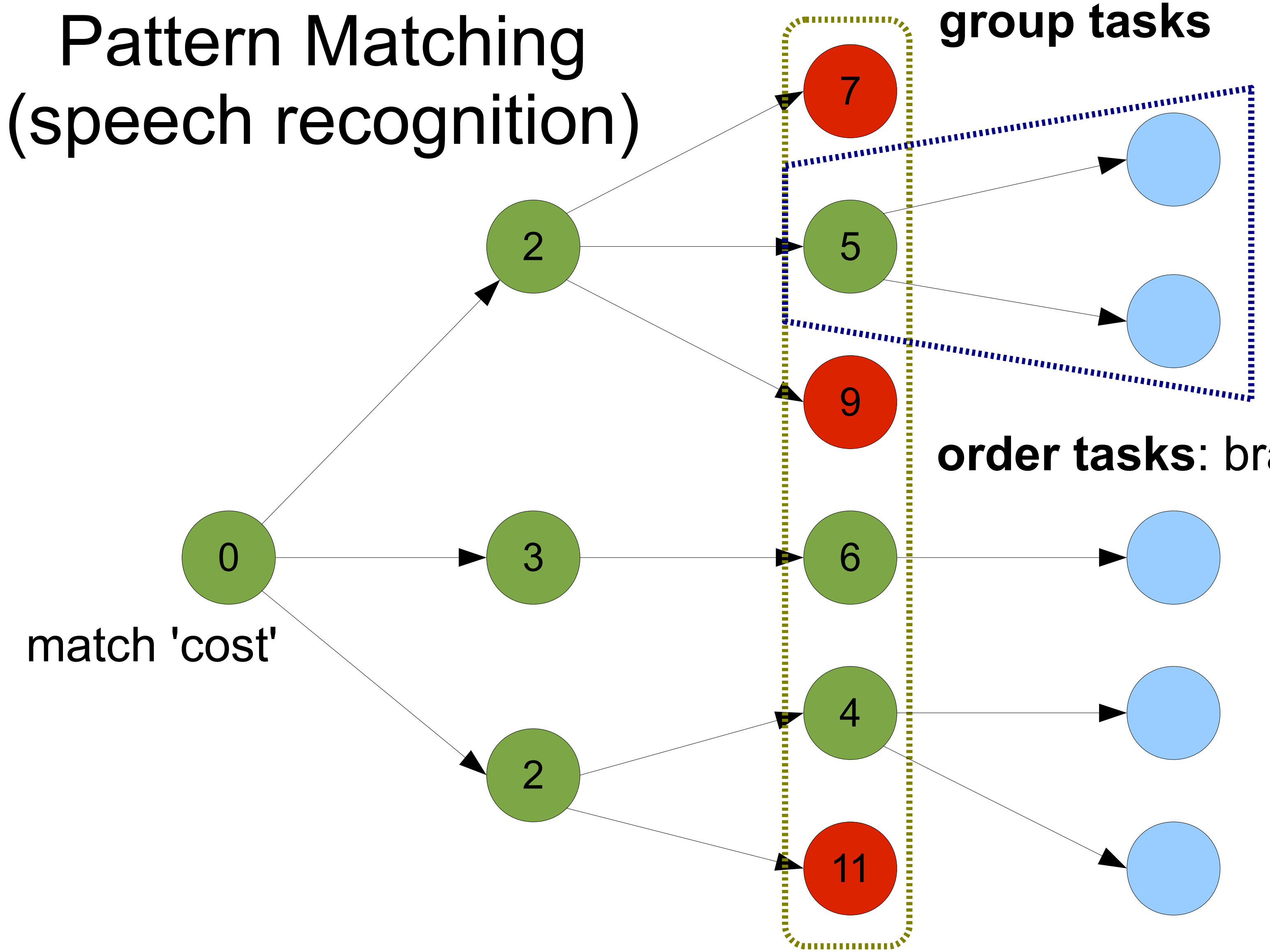
Finding Relationships between Concurrent Tasks



University of BRISTOL

Dependency Analysis

Pattern Matching
(speech recognition)

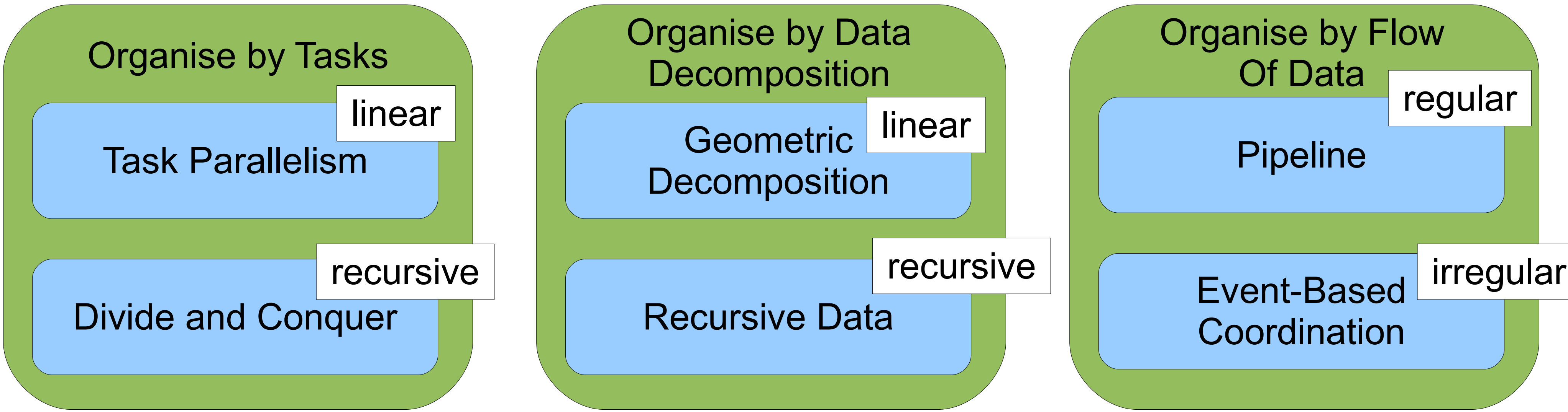


group tasks

data sharing:
- queue of paths
- legal branches
- acoustic i/p

order tasks: branch..& bound

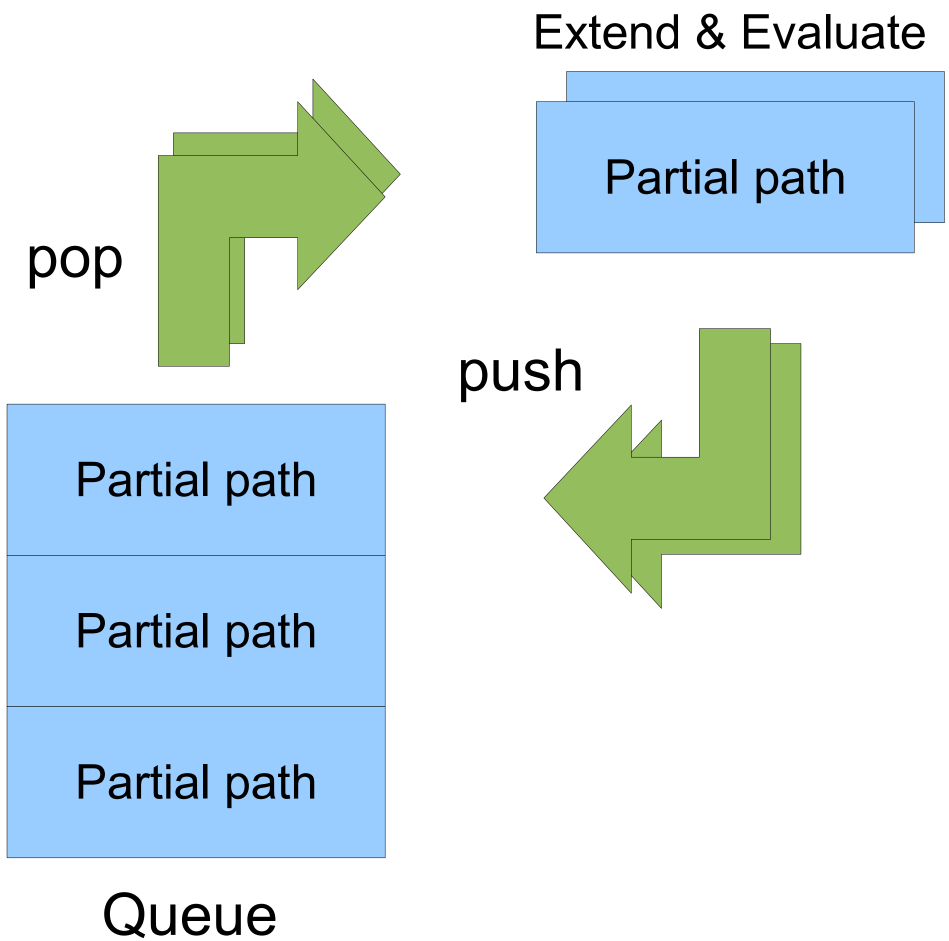
Algorithm Structure



Organise by Tasks - Some Considerations

- No dependencies between tasks
  - massively parallel (vs. embarrassingly serial!)
- Dependencies between tasks
  - Temporal (e.g. speech: real-time constraints)
  - Separable – 'reductions' (we'll see later)
- Cost of setting up task vs. amount of work done
  - See thresholds to switch to serial work (we'll see this in e.g. quicksort)

Organise by Tasks - Task Parallelism

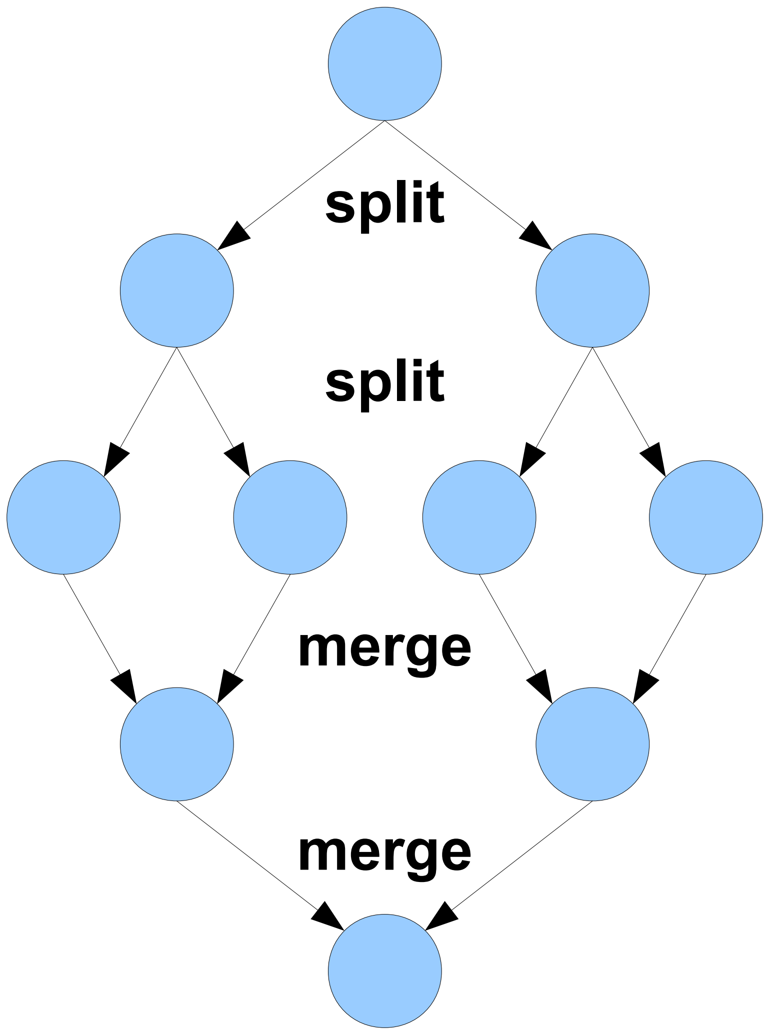


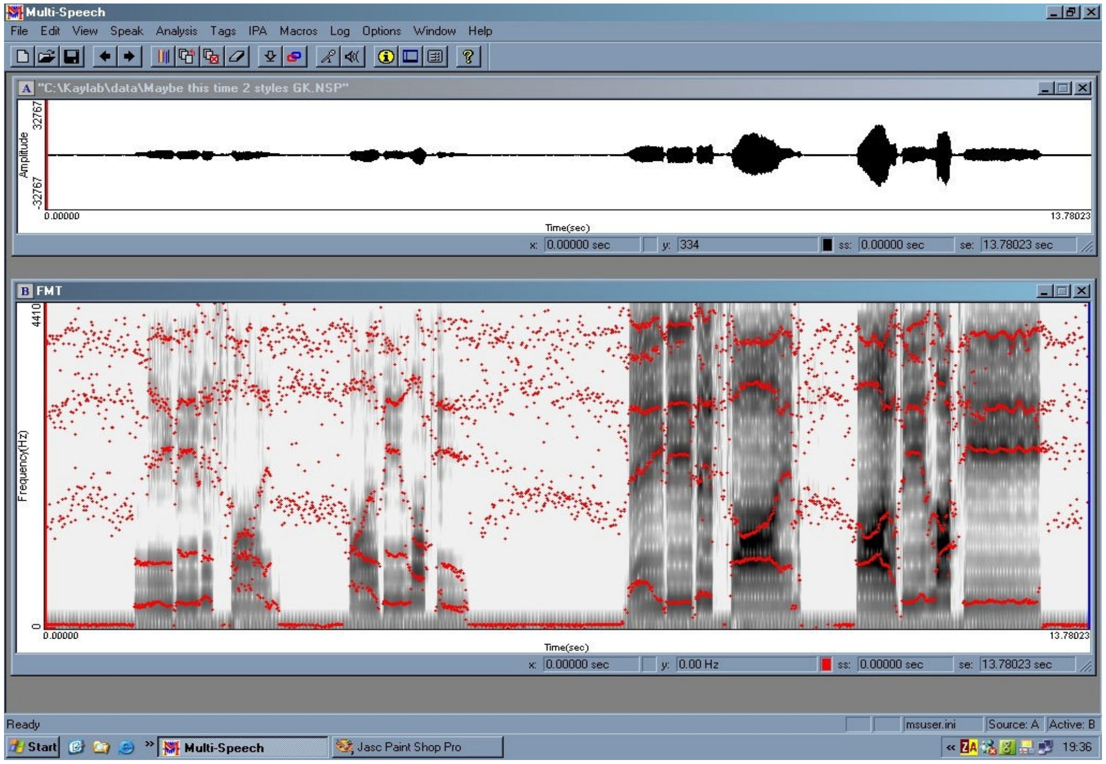
<u>Shared data:</u>
Queue of partial paths
Bounding criterion

Branch & bound implemented with a shared queue

University of BRISTOL

Organise by Tasks - Divide and Conquer

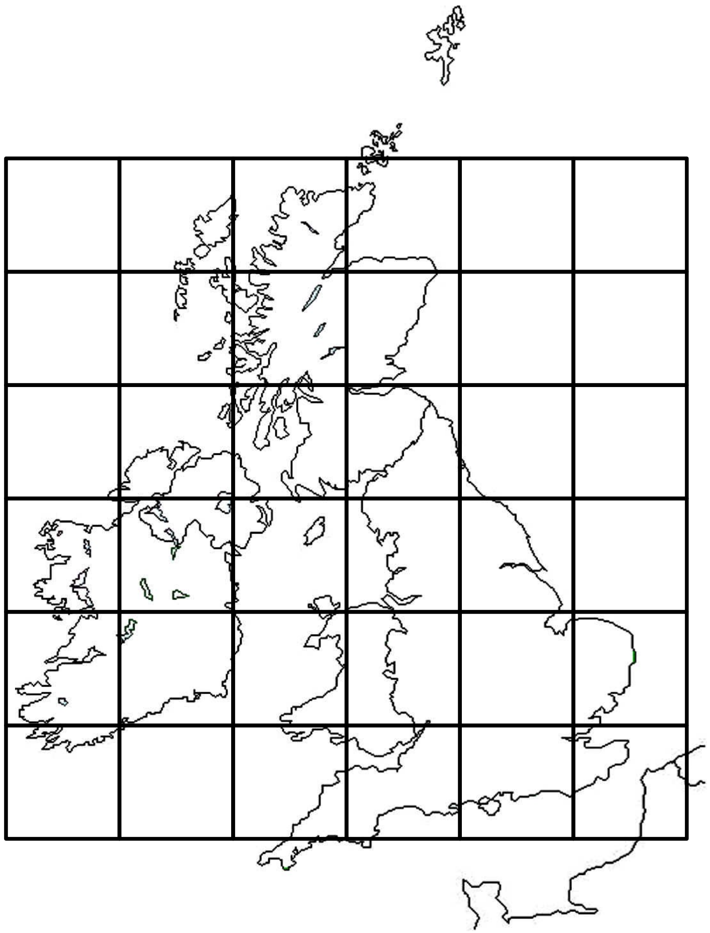




e.g. FFT for speech recognition
Sorting algorithms

University of BRISTOL

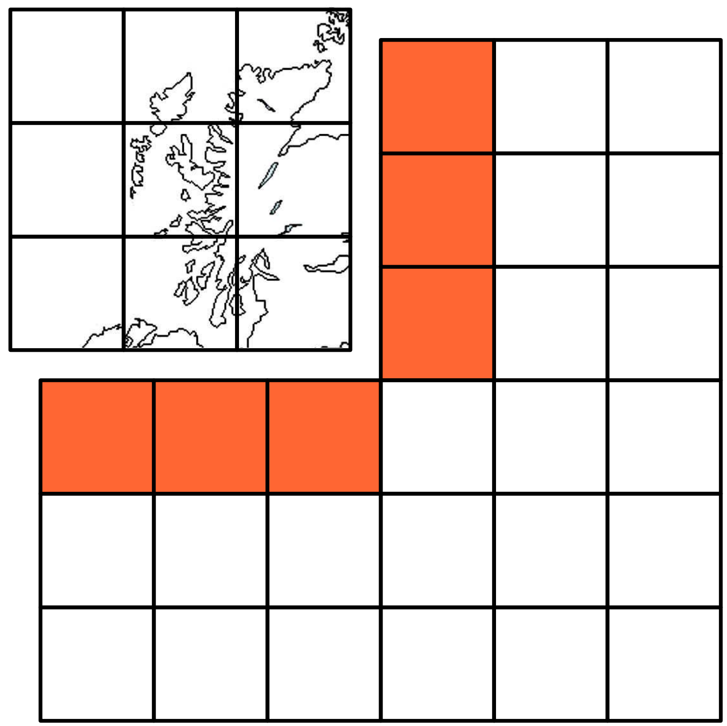
Organise by Data Decomposition - Geometric Decomposition



Grid Cells

Organise by Data Decomposition - Geometric Decomposition

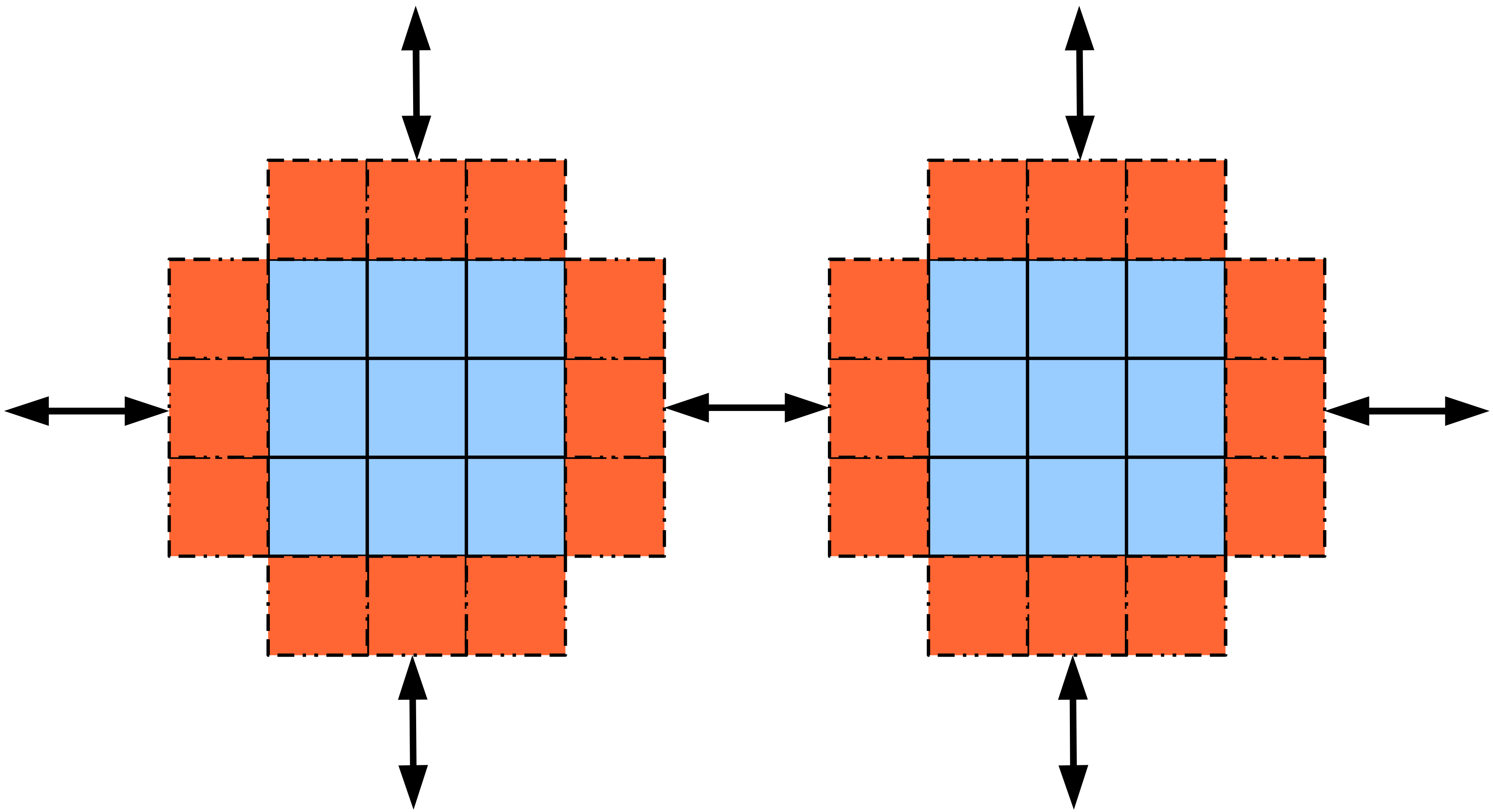
cells assigned to single processor



Exchange local data with neighbours

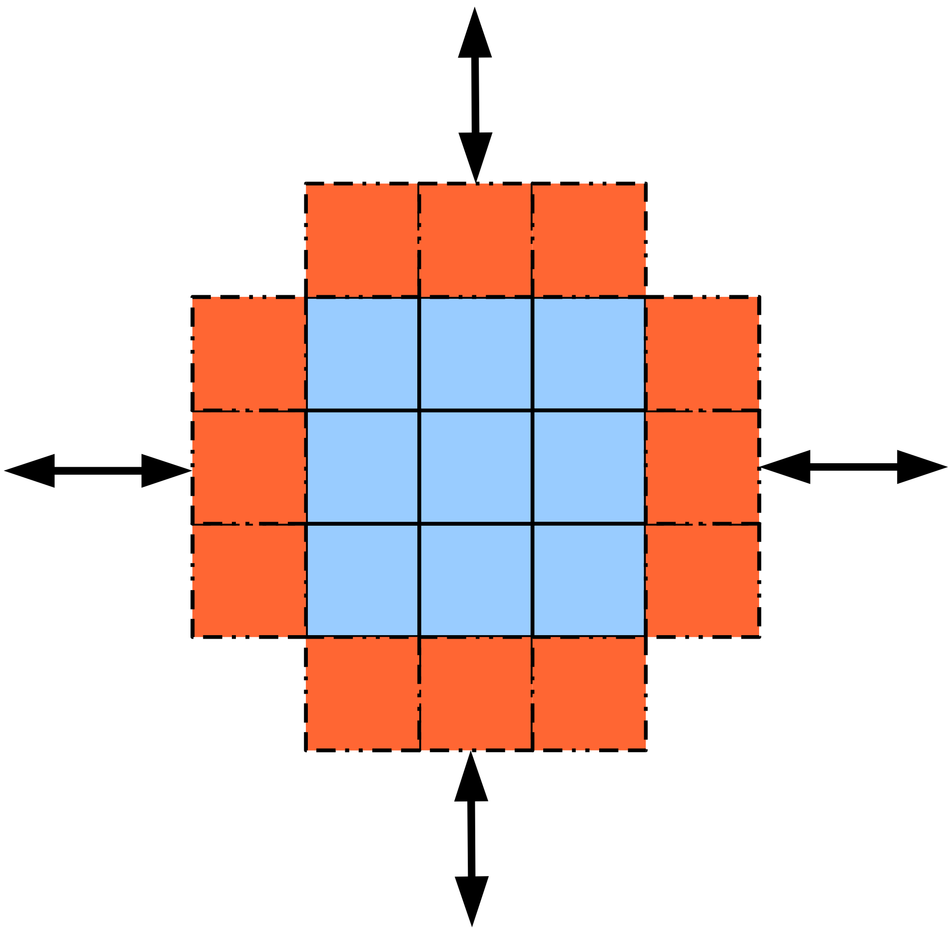
University of BRISTOL

Organise by Data Decomposition - Geometric Decomposition



Halo exchange

Organise by Data Decomposition - Geometric Decomposition



Benefits of halo exchange:

1. Can overlap communication & computation
2. Compute scales with volume but communication scales with surface area

Q. What's wrong with the number of grid cells here?

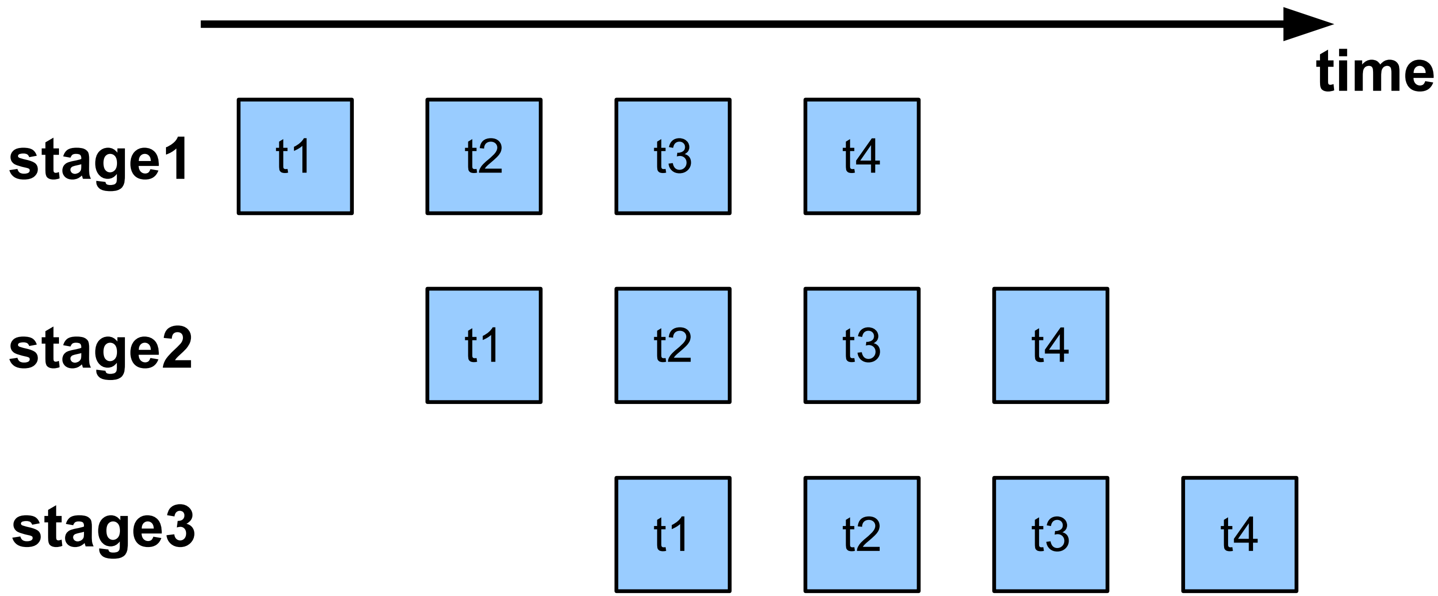
Bonus Q. Any issues with the grid on a globe? Any solutions?

University of BRISTOL

Pipeline

University of BRISTOL

Pipeline

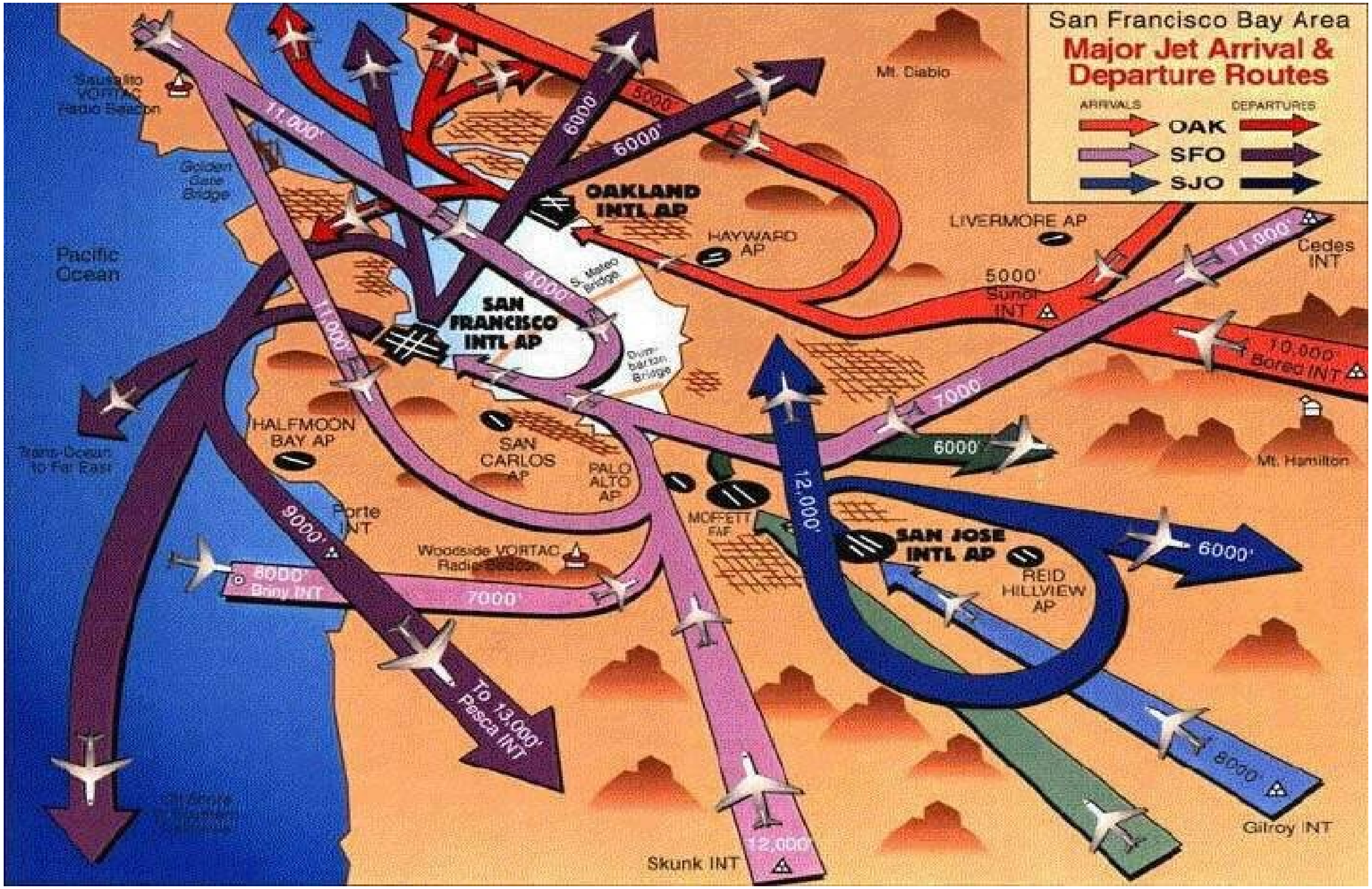


<u>Speech recognition:</u>

1. Discrete Fourier Transform (DFT)
2. manipulation e.g. log
3. Inverse DFT
4. Truncate 'Cepstrum' ..

University of BRISTOL

Event-Based Coordination



Irregular events, ordering constraints (queues can be handy)

University of BRISTOL

Supporting Structures

Program Structures	Program Structures
SPMD	Shared Data
Master/Worker	Shared Queue
Loop Parallelism	Distributed Array
Fork/Join	

Useful **idioms** rather than unique implementations

Single Program Multiple Data

rank = 0

```
if(rank == 0) {
printf("MASTER\n");
}
else {
printf("OTHER\n");
}
```

rank = 1

```
if(rank == 0) {
printf("MASTER\n");
}
else {
printf("OTHER\n");
}
```

- Only one program to manage
- Conditionals based on thread or process IDs
- Load balance predictable (implicit in branching)
- Plenty of examples and practice when we look at MPI

University of BRISTOL

Master/Worker

Use when load balance is not predictable..

..& work cannot be distributed using loops

- PEs may have different capabilities
- A bag of independent tasks is ideal
- Workers take from bag, process, then take another

Load is **automatically** balanced in this way

Master/Worker (Cloud): MapReduce

<u>Big Data:</u>
Google Processed 20PB/day in 2008 using MapReduce
Also used by Yahoo, FaceBook, eBay etc.

Loop Parallelism

Use if computational expense is concentrated in loops (common in scientific code)

1. Profile code to find 'hot-spots'
2. Eliminate dependencies between iterations (e.g. private copies & reductions)
3. Parallelise loops (easy in OpenMP)
4. Tune the performance, e.g. via scheduling

We'll get plenty of practice with OpenMP

Fork/Join

Use if the number of concurrent tasks varies, e.g. if tasks are created recursively

- Beware: overhead of creating a new UEs (Uinits of Execution, e.g. thread or process)
  - Direct vs. indirect mappings from tasks to Ues
- Sorting algos are an examples

University of BRISTOL

Shared Data

- Try to avoid, as can limit scalability
- Use a concurrency-controlled (e.g. 'thread-safe') data type:
  - One-at-a-time: critical region/'mutex'
  - Look for non-interfering operations e.g. readers vs. writers
  - If pushed, finer grained critical regions, but this will increase complexity & hence the chance of a bug
- 'Shared Queue' is an instance of 'Shared Data'

Distributed Arrays

In a nutshell: partition data and distribute so that data is close to computation.

- Why? Memory access (esp. over a network) is slow relative to computation.
- Simple concept but the devil is in the details
- Some terminology:
  - 1D block, 2D block and block cyclic distribution

Libraries: e.g. ScaLAPACK

University of BRISTOL

Recap of Key Points

Design..

- for massively parallel systems
- because if not today they will be tomorrow
- and in all areas of computing

Design Patterns..

- provide useful – recurring - solutions
- & structure to the process

Implementation Mechanisms

OpenMP & Pthreads

MPI

OpenCL