LEIDEN UNIVERSITY MEDICAL CENTER

Clusters and Parallelising Workflows

Jeroen F. J. Laros

Leiden Genome Technology Center

Department of Human Genetics

Center for Human and Clinical Genetics

Introduction

Sequencers: Ion Torrent

Figure 1: Ion torrent.

Characteristics:

- Moderate throughput.
- Single end (for now).
- High accuracy.
- Read length $\pm$200bp.
- Short run time.
- Cheap runs.

Introduction

Sequencers: HiSeq

Figure 2: HiSeq 2000.

Characteristics:

- High throughput.
- Paired end.
- High accuracy.
- Read length 2×150bp.
- Relatively long run time.
- Relatively expensive.

Introduction

Raw data analysis

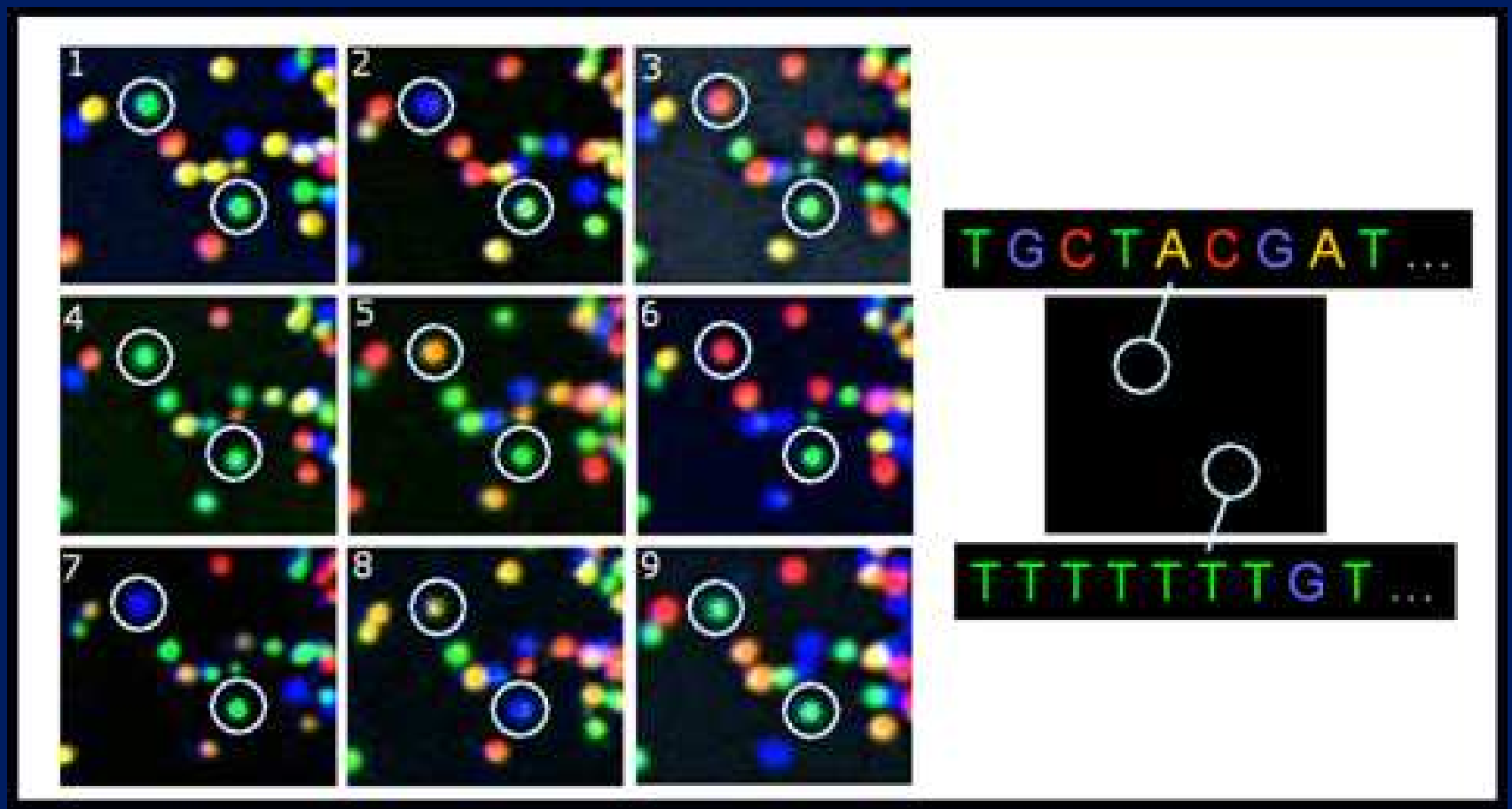

Figure 3: Base calling.

Introduction

Result of base calling

```
1  @FCC0ADRACXX:2:1101:1684:1875#GCGGAACT/1
2  AGGGGATGCAACCTCGAGGAGGAAAGGAACGAAAGAGGAAGGGAG...
3  +
4  BS\ceeeeggggggiihfhighihiiiihiihhhihfhighiiiii...
5  @FCC0ADRACXX:2:1101:1714:1885#GCGGAACT/1
6  ATTTTCTCAATGTCTACCACTGCGGGTAACACTTTGTGTTTCCCA...
7  +
8  bbbeeeeeggggggiihiiiiiiiiiighiiiiiiiiggfghiihi...
```

Listing 1: FastQ file.

These files are around 1T big.

Introduction

Some figures

Runtime 2 weeks (5-10 full genomes).

Produces 6T of pictures per flowcell, 500G base calling.

Needs continuous connection to the storage, otherwise it stalls.

Introduction

Some figures

Runtime 2 weeks (5-10 full genomes).

Produces 6T of pictures per flowcell, 500G base calling.

Needs continuous connection to the storage, otherwise it stalls.

Basecalling:

24		time in hours for base calling per flowcell
12	$\times$	number of cores
2	$\times$	number of flowcells
576		CPU hours per run

On a desktop machine this would take 48 days.

Introduction

Data analysis

Resequencing pipelines can roughly be divided in five steps.

Introduction

Data analysis

Resequencing pipelines can roughly be divided in five steps.

1. Pre-alignment.
    - Quality control.
    - Data cleaning.

Introduction

Data analysis

Resequencing pipelines can roughly be divided in five steps.

1. Pre-alignment.
    - Quality control.
    - Data cleaning.
2. Alignment.
    - Post-alignment quality control.

Introduction

Data analysis

Resequencing pipelines can roughly be divided in five steps.

1. Pre-alignment.
    - Quality control.
    - Data cleaning.
2. Alignment.
    - Post-alignment quality control.
3. Variant calling.

Introduction

Data analysis

Resequencing pipelines can roughly be divided in five steps.

1. Pre-alignment.
    - Quality control.
    - Data cleaning.
2. Alignment.
    - Post-alignment quality control.
3. Variant calling.
4. Filtering.
    - Post-variant calling quality control.

Introduction

Data analysis

Resequencing pipelines can roughly be divided in five steps.

1. Pre-alignment.
   - Quality control.
   - Data cleaning.
2. Alignment.
   - Post-alignment quality control.
3. Variant calling.
4. Filtering.
   - Post-variant calling quality control.
5. Annotation.

Introduction

Large projects

Genome of the Netherlands:

- 750 full genomes.

Introduction

Large projects

Genome of the Netherlands:

- 750 full genomes.

Centre for Genome Diagnostics:

- 10 full genomes.

Introduction

Large projects

Genome of the Netherlands:

- 750 full genomes.

Centre for Genome Diagnostics:

- 10 full genomes.

Geuvadis:

- QC for 667 samples done in two days.

These types of analysis would be impossible without a cluster.

Introduction

Pipelines

To automate these procedures, we usually make a *pipeline*.

```
1 bwa aln ref.fa s_1.fq > s_1.sai
2 bwa aln ref.fa s_2.fq > s_2.sai
3 bwa sampe ref.fa s_1.sai s_2.sai s_1.fq s2_fq > s.sam
4 samtools view -bt ref.fa -o s.bam s.sam
5 samtools sort s.bam s.sorted
6 samtools mpileup -f ref.fa s.sorted.bam > s.pileup
```

Listing 2: Bash script snippet.

These scripts are *linear*.

Introduction

Pipelines

To automate these procedures, we usually make a *pipeline*.

```
1 bwa aln ref.fa s_1.fq > s_1.sai
2 bwa aln ref.fa s_2.fq > s_2.sai
3 bwa sampe ref.fa s_1.sai s_2.sai s_1.fq s2_fq > s.sam
4 samtools view -bt ref.fa -o s.bam s.sam
5 samtools sort s.bam s.sorted
6 samtools mpileup -f ref.fa s.sorted.bam > s.pileup
```

Listing 2: Bash script snippet.

These scripts are *linear*.

Note that line 1 and line 2 could be run simultaneous.

Using servers

Logging in

There are lots of ways to connect to a server.

- HTTP – When visiting websites.
- IMAP – When fetching mail.
- ...

In order to execute commands, we need to *log in*.

Using servers

Logging in

There are lots of ways to connect to a server.

- HTTP – When visiting websites.
- IMAP – When fetching mail.
- ...

In order to execute commands, we need to *log in*.

We use a *secure* protocol to log in.

- Most plain text protocols are blocked by firewalls.
- When working with patient data, we don't want eavesdropping.
- The connection from your machine to the server is *encrypted*.

Using servers

Secure Shell

Using Secure Shell (ssh):

```
ssh user@host
```

Keyword	Explanation.
user	Your *username* on the *server*.
host	Name of the *server*.

Table 1: Basic **ssh** usage.

```
1 ssh jfjlaros@shark.lumc.nl
```

Listing 3: Example login.

File transfer

Copying data

We frequently need to transfer data before and after we do an analysis.

- The input needs to be on the server.
- The output needs to be copied back.

File transfer

Copying data

We frequently need to transfer data before and after we do an analysis.

- The input needs to be on the server.
- The output needs to be copied back.

We also use a secure protocol to copy.

- If Secure Shell works, then this will work too (same protocol).
- Two way traffic.
  - Copy data from your machine to the server (uploading).
  - Copy data from the server to your machine (downloading).

File transfer

Secure Copy

Copying something to the server:

```
scp localfile user@host:/path/to/remotefile
```

Keyword	Explanation.
`localfile`	Name of the file on *your* computer.
`user`	Your *username* on the *server*.
`host`	Name of the *server*.
`/path/to/`	Directory on the *server* to store the file.
`remotefile`	Name of the file on the *server*.

Table 2: Basic `scp` usage.

```
1 scp myfile jfjlaros@shark.lumc.nl:/tmp/yourfile
```

Listing 4: Example secure file transfer.

File transfer

Secure Copy

Some defaults (when left empty):

```
scp localfile user@host:/path/to/remotefile
```

Keyword	Explanation.
`user`	The *local* username.
`/path/to/`	The home directory on the server.
`remotefile`	The same as the name of the local file.
`localfile`	May be replaced by a “ .”

Table 3: `scp` defaults.

```
1 scp myfile shark.lumc.nl:
2 scp shark.lumc.nl:myfile .
```

Listing 5: Example secure file transfer.

Connecting from an other OS

Windows

Windows does not have the **`ssh`** command, but there are programs that give the same functionality.

PuTTY – A Free Telnet/SSH Client.

A software package containing (amongst others):

- PuTTY: Secure Shell client.
- PSCP: Secure Copy client.
- More related tools available on the website.

http://www.chiark.greenend.org.uk/~sgtatham/putty

Connecting from an other OS

Windows

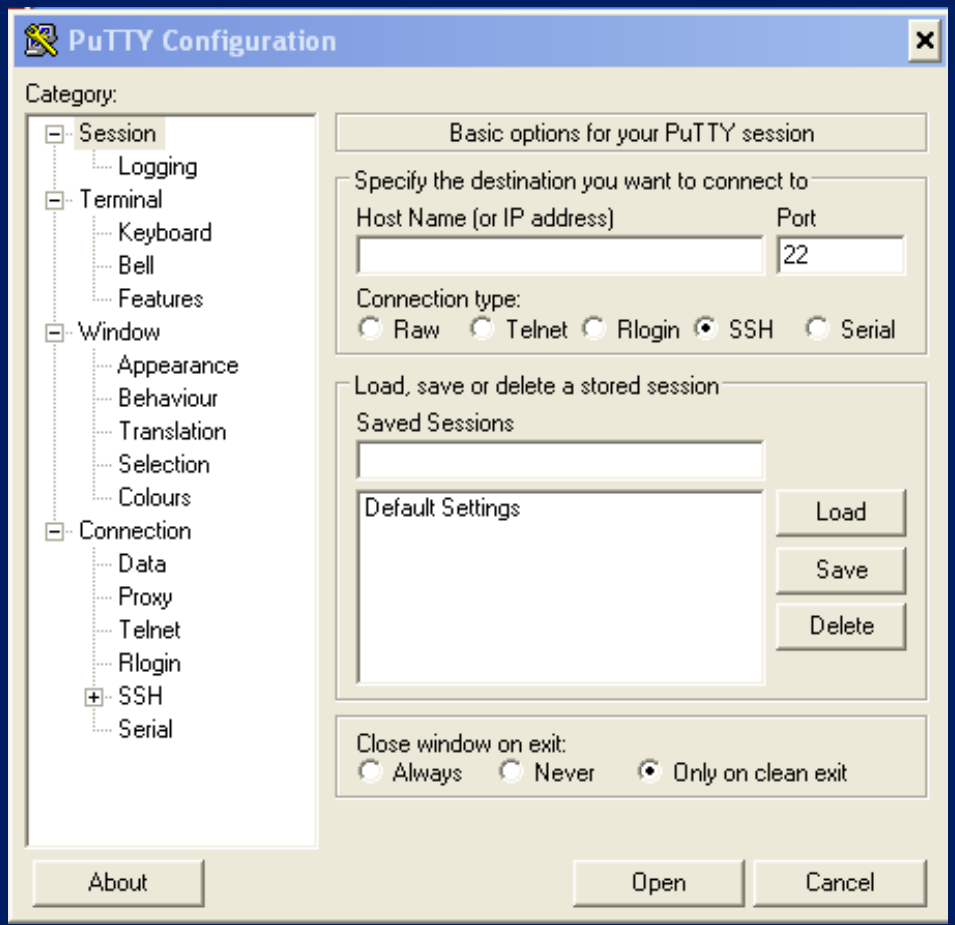

Figure 4: Connecting to a server using PuTTY.

Connecting from an other OS

Windows

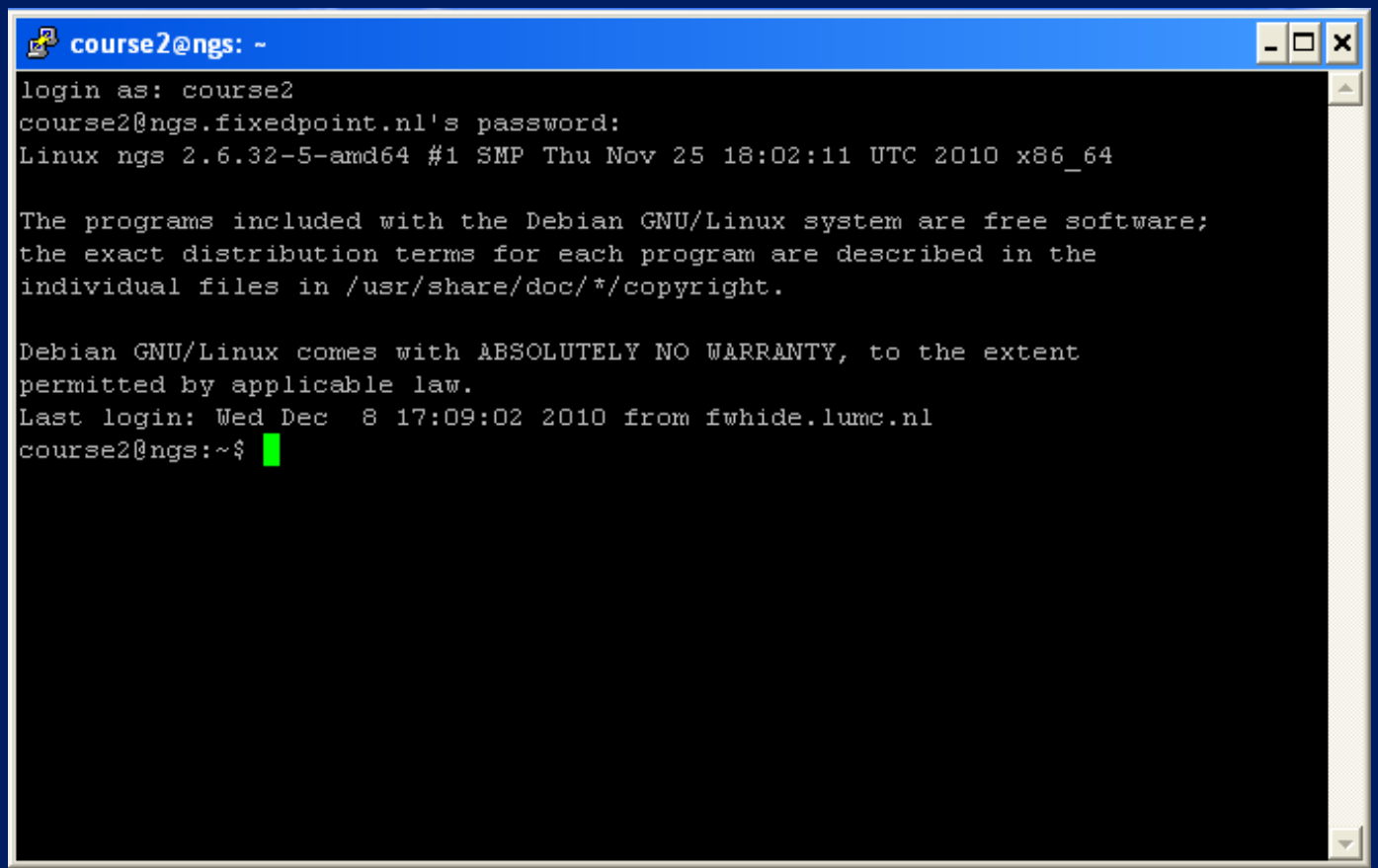

Figure 5: A terminal when connected to a server.

Examples

Typical workflow

When doing an analysis, the general workflow looks like this:

- First copy the input data to the server.
- Log on to the server.
- Run the analysis remotely.
- Copy the results from the server.
- Clean up the input data and the results on the server.
- Log out.

Examples

Typical workflow: an example

Step one: preparing the input.

On your machine, copy the raw data to the server, then log in on the server.

```
1 scp reads.fq jfjlaros@shark.lumc.nl:
2 ssh jfjlaros@shark.lumc.nl
```

Listing 6: Copy and login.

Now the file **`reads.fq`** is available on the server.

Examples

Typical workflow: an example

Step two: The analysis.

On the server, you can do an analysis.

```
1 bwa aln ref.fa s_1.fq > s_1.sai
2 bwa aln ref.fa s_2.fq > s_2.sai
3 bwa sampe ref.fa s_1.sai s_2.sai s_1.fq s2_fq > s.sam
4 samtools view -bt ref.fa -o s.bam s.sam
5 samtools sort s.bam s.sorted
6 samtools mpileup -f ref.fa s.sorted.bam > s.pileup
```

Listing 7: Bash script snippet.

Examples

Typical workflow: an example

Step three: Retrieving the output.

Copy the output from the server back to your own machine.

```
1 scp jfjlaros@shark.lumc.nl:reads.pileup .
```

Listing 8: Copy the result.

Examples

Typical workflow: an example

Step three: Retrieving the output.

Copy the output from the server back to your own machine.

```
1 scp jfjlaros@shark.lumc.nl:reads.pileup .
```

Listing 8: Copy the result.

Step four: Cleaning up.

Clean up on the server and leave.

```
1 rm reads.*
2 logout
```

Listing 9: Clean up and leave.

Why remote servers?

Clusters

Figure 6: Dell M610 blade server.

Why remote servers?

Clusters

Massive parallel computing.

- A large number of computers working together.
- Analyse lots of samples at the same time.
- Sometimes a way to reduce memory requirements (if the problem permits it).
- Very suitable for NGS, especially alignment.

Why remote servers?

Clusters

Massive parallel computing.

- A large number of computers working together.
- Analyse lots of samples at the same time.
- Sometimes a way to reduce memory requirements (if the problem permits it).
- Very suitable for NGS, especially alignment.

Cons:

- Not all problems are suitable for parallel computation.
- Programs must be adjusted to make use of a cluster.
    - Chop the problem up in parts / combine the results.

Why remote servers?

Clusters

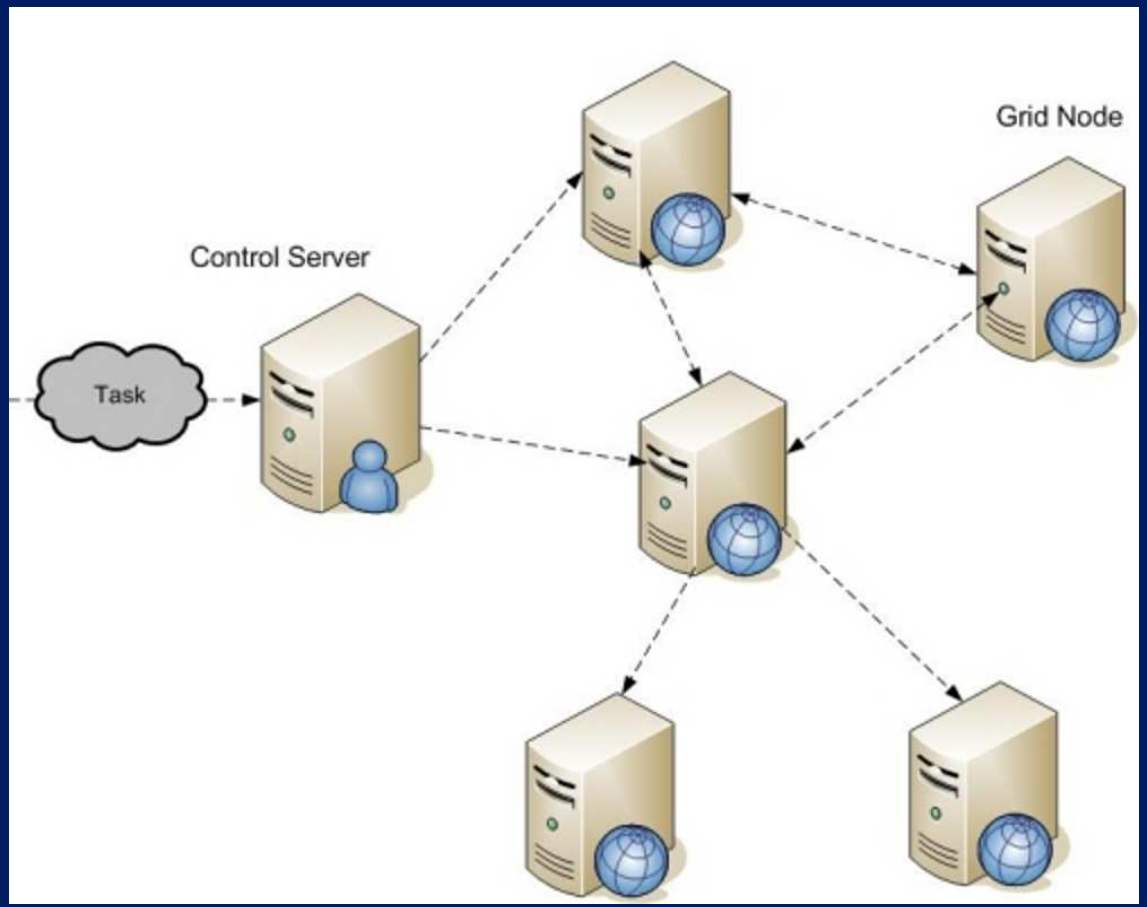

Figure 7: Schematic overview of a cluster.

Why remote servers?

Clusters

General characteristics of a cluster.

- Jobs are submitted to a *control node*.
- The control node dispatches a job to a free *worker node*.
- Jobs are monitored.
  - If a worker node doesn't finish for some reason, the job gets dispatched to an other worker node.
  - If all worker nodes are finished, the control node can alert the user that his jobs are finished.
- Jobs can be prioritised.
- ...

Parallelisation

When can we use it?

"No program can run more quickly than the longest chain of dependent calculations."

Parallelisation

When can we use it?

"No program can run more quickly than the longest chain of dependent calculations."

An algorithm can be parallelised if it has independent components:

- A sub-result is not dependent on an other sub-result.

Parallelisation

When can we use it?

"No program can run more quickly than the longest chain of dependent calculations."

An algorithm can be parallelised if it has independent components:

- A sub-result is not dependent on an other sub-result.

Parallelisable problems:

- Alignment.

Non-parallelisable problems:

- De novo assembly.

Parallelisation

How to distribute a problem

Parallelise the algorithm itself.

- Requires advanced programming skills.
- Requires a firm understanding of the algorithm.

Parallelisation

How to distribute a problem

Parallelise the algorithm itself.

- Requires advanced programming skills.
- Requires a firm understanding of the algorithm.

Split the input and combine the outputs.

- Requires a bit of understanding of the algorithm.

Parallelisation

How to distribute a problem

Parallelise the algorithm itself.

- Requires advanced programming skills.
- Requires a firm understanding of the algorithm.

Split the input and combine the outputs.

- Requires a bit of understanding of the algorithm.

Run the whole pipeline in parallel.

- Most widely used method.
- Only possible if you have a large number of samples.

Parallelisation

Complexity of alignment

$$\mathcal{O}(m \cdot \log_2(n))$$

- m Number of reads.
- n Length of the reference sequence.

Parallelisation

Complexity of alignment

$$\mathcal{O}(m \cdot \log_2(n))$$

m	Number of reads.
n	Length of the reference sequence.

Example $m = 100, n = 100$:

$$\begin{aligned} t &= 100 \cdot \log_2(100) \\ &= 100 \cdot 7 \end{aligned}$$

Parallelisation

Choices in parallelisation

Split the reference sequence $m = 100, n = 50$:

$$\begin{aligned} t &= 100 \cdot \log_2(50) \\ &= 100 \cdot 6 \end{aligned}$$

This only makes it $1.167\times$ faster.

Parallelisation

Choices in parallelisation

Split the reference sequence $m = 100, n = 50$:

$$\begin{aligned} t &= 100 \cdot \log_2(50) \\ &= 100 \cdot 6 \end{aligned}$$

This only makes it $1.167\times$ faster.

Split the reads $m = 50, n = 100$:

$$\begin{aligned} t &= 50 \cdot \log_2(100) \\ &= 50 \cdot 7 \end{aligned}$$

This makes it $2\times$ faster.

Parallelisation

Choices in parallelisation (2)

An other reason for not splitting the reference sequence:

Results will change:

- Reads will be mapped with more errors in the wrong place.
- Multiple alignments.

Parallelisation

Choices in parallelisation (2)

An other reason for not splitting the reference sequence:

Results will change:

- Reads will be mapped with more errors in the wrong place.
- Multiple alignments.

So, don't be tempted to align to chromosomes independently.

Parallelisation

Choices in parallelisation (2)

An other reason for not splitting the reference sequence:

Results will change:

- Reads will be mapped with more errors in the wrong place.
- Multiple alignments.

So, don't be tempted to align to chromosomes independently.

In general, check whether your results change if you distribute a problem.

Pause

Using a cluster

Sun Grid Engine

A cluster needs a job scheduler.

- Open source batch-queuing system developed by Sun Microsystems.
- Now owned by Oracle, no longer free.
- Fully compatible alternatives:
  - Son of Grid.
  - Open Grid Scheduler.
  - Univa Grid Engine (commercial).

http://gridscheduler.sourceforge.net

Using a cluster

Sun Grid Engine

A cluster needs a job scheduler.

- Open source batch-queuing system developed by Sun Microsystems.
- Now owned by Oracle, no longer free.
- Fully compatible alternatives:
  - Son of Grid.
  - Open Grid Scheduler.
  - Univa Grid Engine (commercial).

http://gridscheduler.sourceforge.net

Using a cluster

Pipelines

Classical pipeline:

- Linear.
- Shell script.
    - Or with the exec() function in your favourite language.

Using a cluster

Pipelines

Classical pipeline:

- Linear.
- Shell script.
    - Or with the exec() function in your favourite language.

Drawbacks:

- Parallelisation is done manually.
- Syncing must be done manually.
- Extensive error handling.
- No save points.
- No dry runs.

Using a cluster

A different viewpoint

What if. . .

- All commands are atomic.
  - We describe input and output.
- We build a *dependency graph*.
- Trace a path in this graph to find a workflow.

Using a cluster

A different viewpoint

What if. . .

- All commands are atomic.
  - We describe input and output.
- We build a *dependency graph*.
- Trace a path in this graph to find a workflow.

This way we do not need to:

- Design a workflow.
- Figure out which parts can be run in parallel.

Theoretical solution

Petri nets

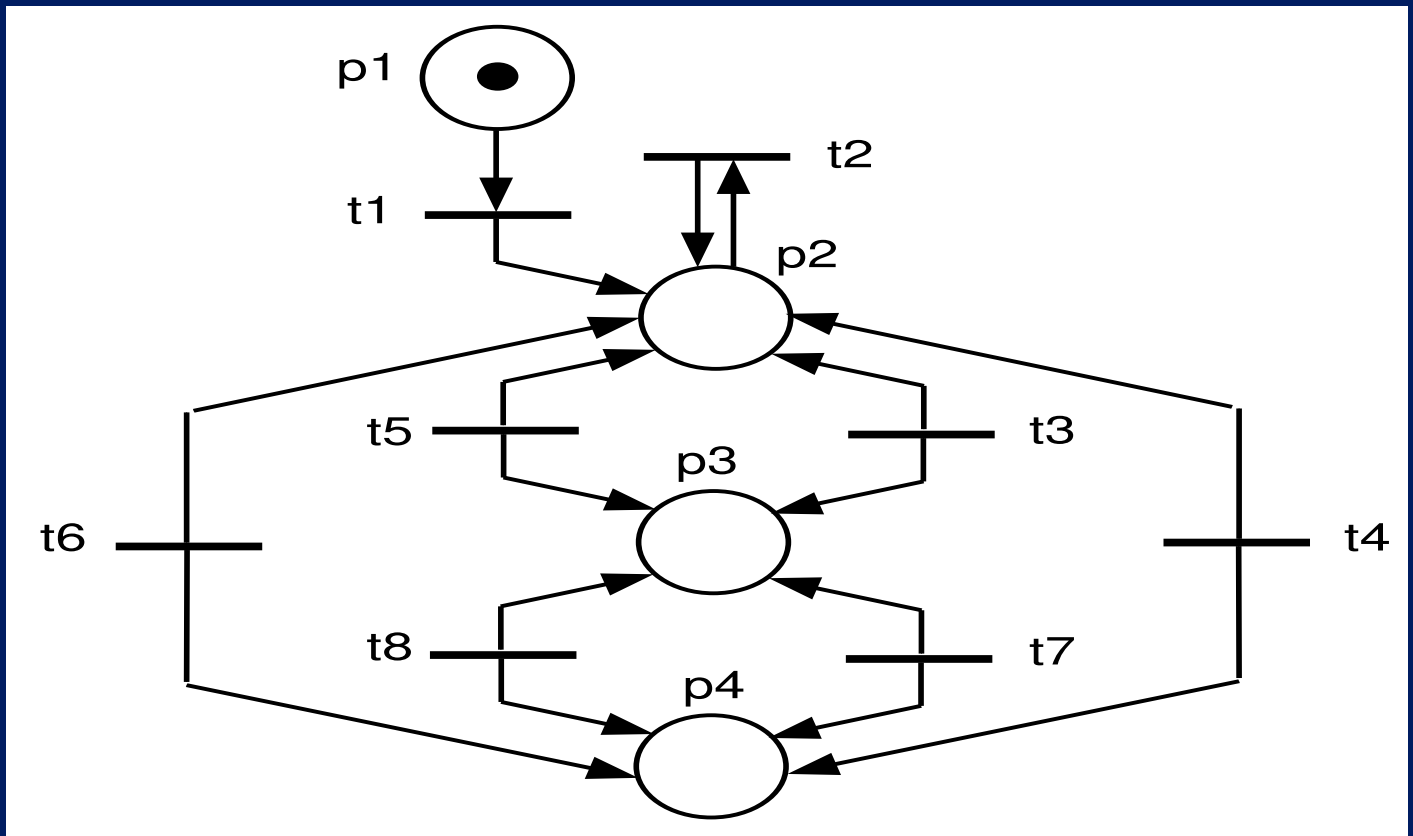

Figure 8: A model for parallelism.

Theoretical solution

Petri nets

"Mathematical modelling language for the description of distributed systems."

Background:

- Described first in 1939 by Carl Adam Petri (age 13).
- Originally intended to describe chemical processes.
- Graphical notation for stepwise processes.
- Choice, iteration, and concurrent execution.

http://en.wikipedia.org/wiki/Petri_net

Theoretical solution

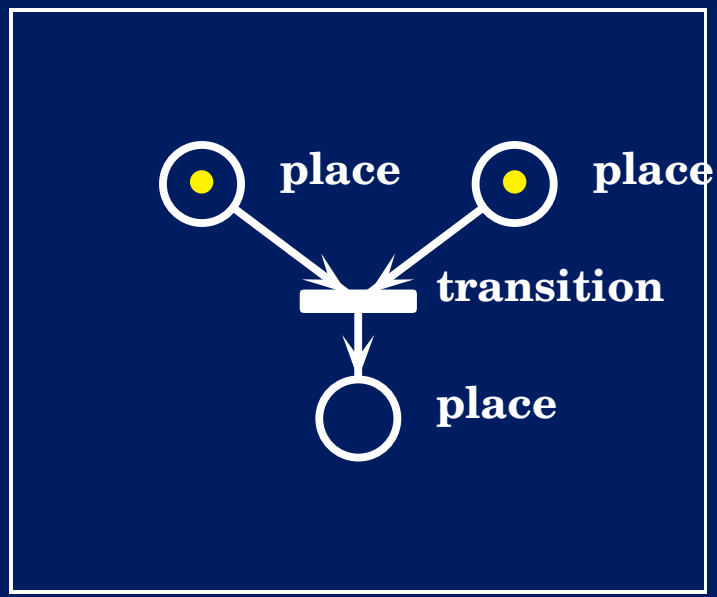

Figure 9: A simple Petri net.

A *transition* has:

- A *preset*: a set of *input places*.
- A *postset*: a set of *output places*.

The output of one transition can be the input of an other.

Theoretical solution

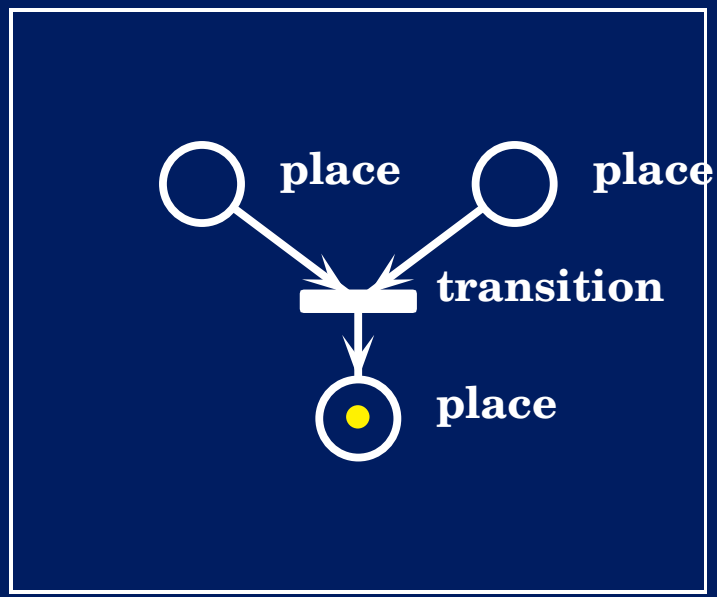

Figure 9: A simple Petri net.

A *transition* has:

- A *preset*: a set of *input places*.
- A *postset*: a set of *output places*.

The output of one transition can be the input of an other.

Theoretical solution

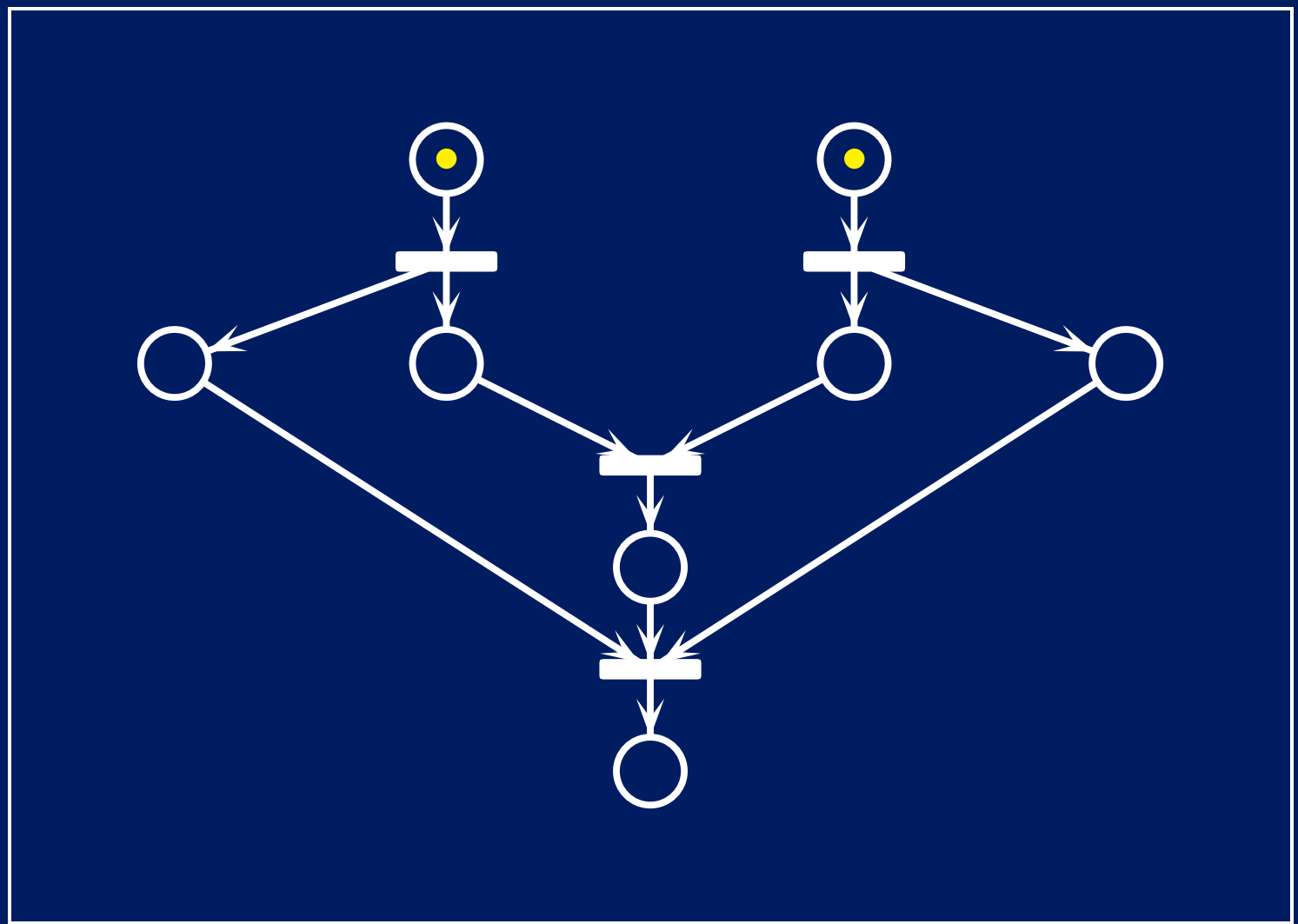

Figure 10: A more complicated Petri net.

Theoretical solution

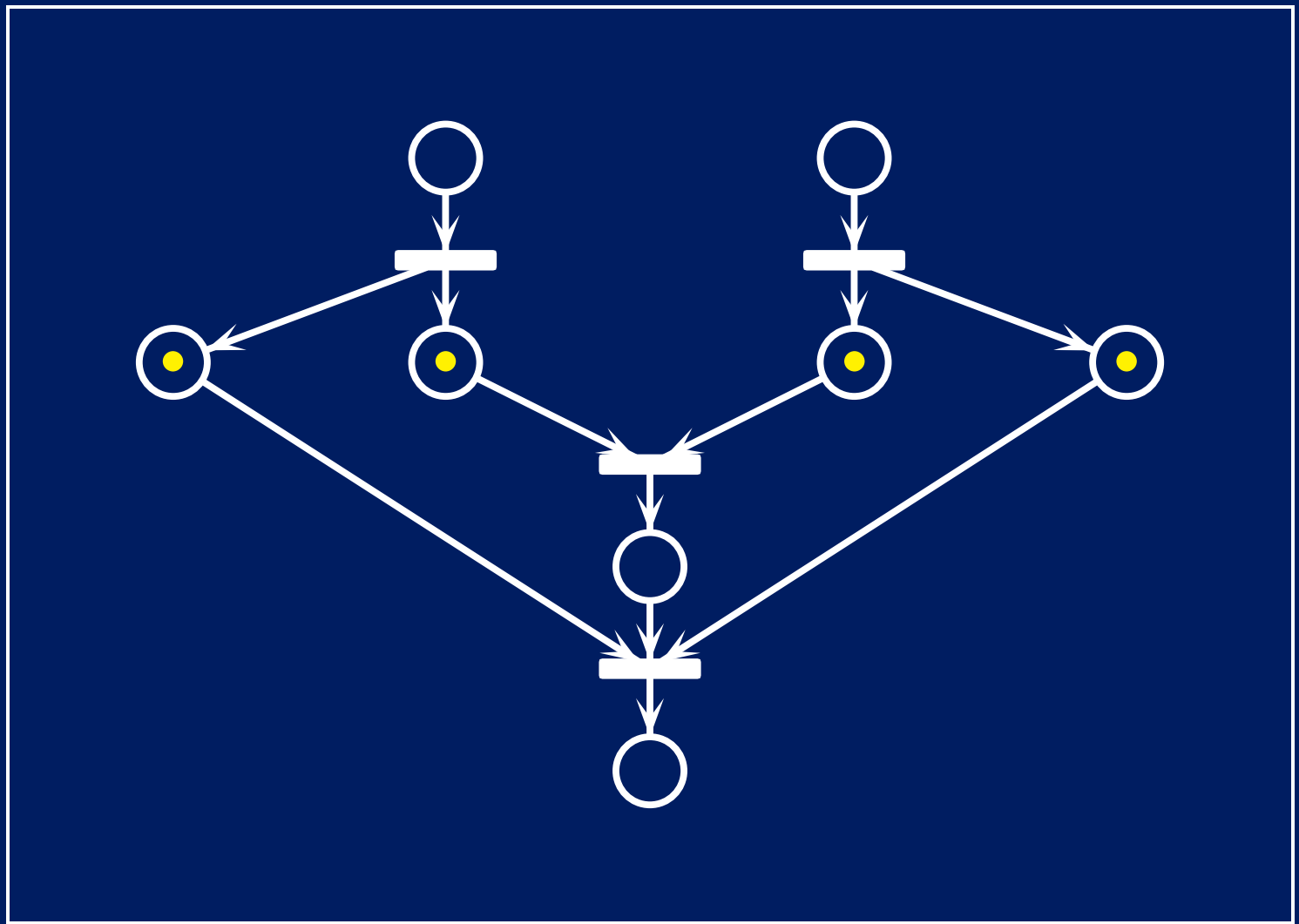

Figure 10: A more complicated Petri net.

Theoretical solution

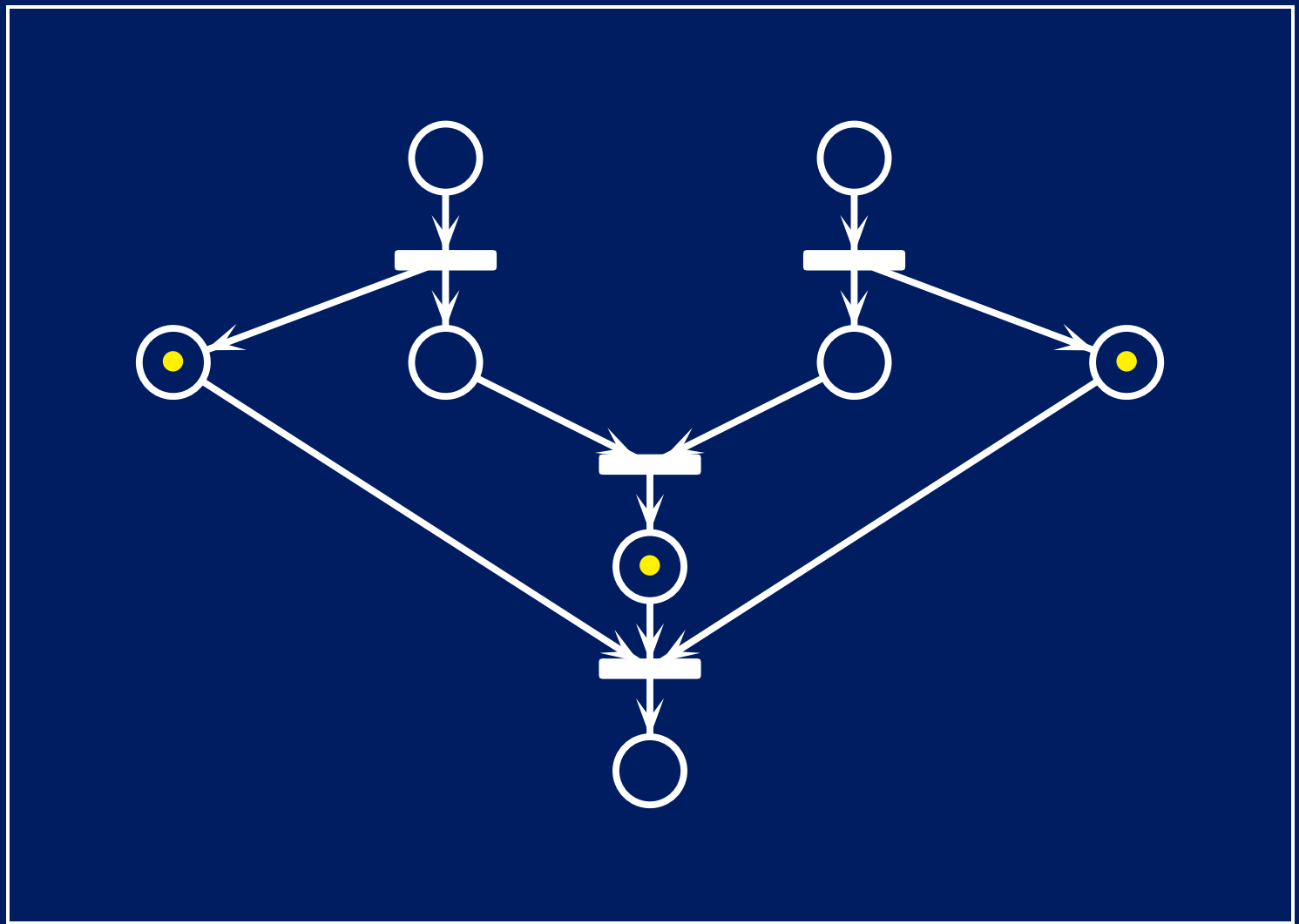

Figure 10: A more complicated Petri net.

Theoretical solution

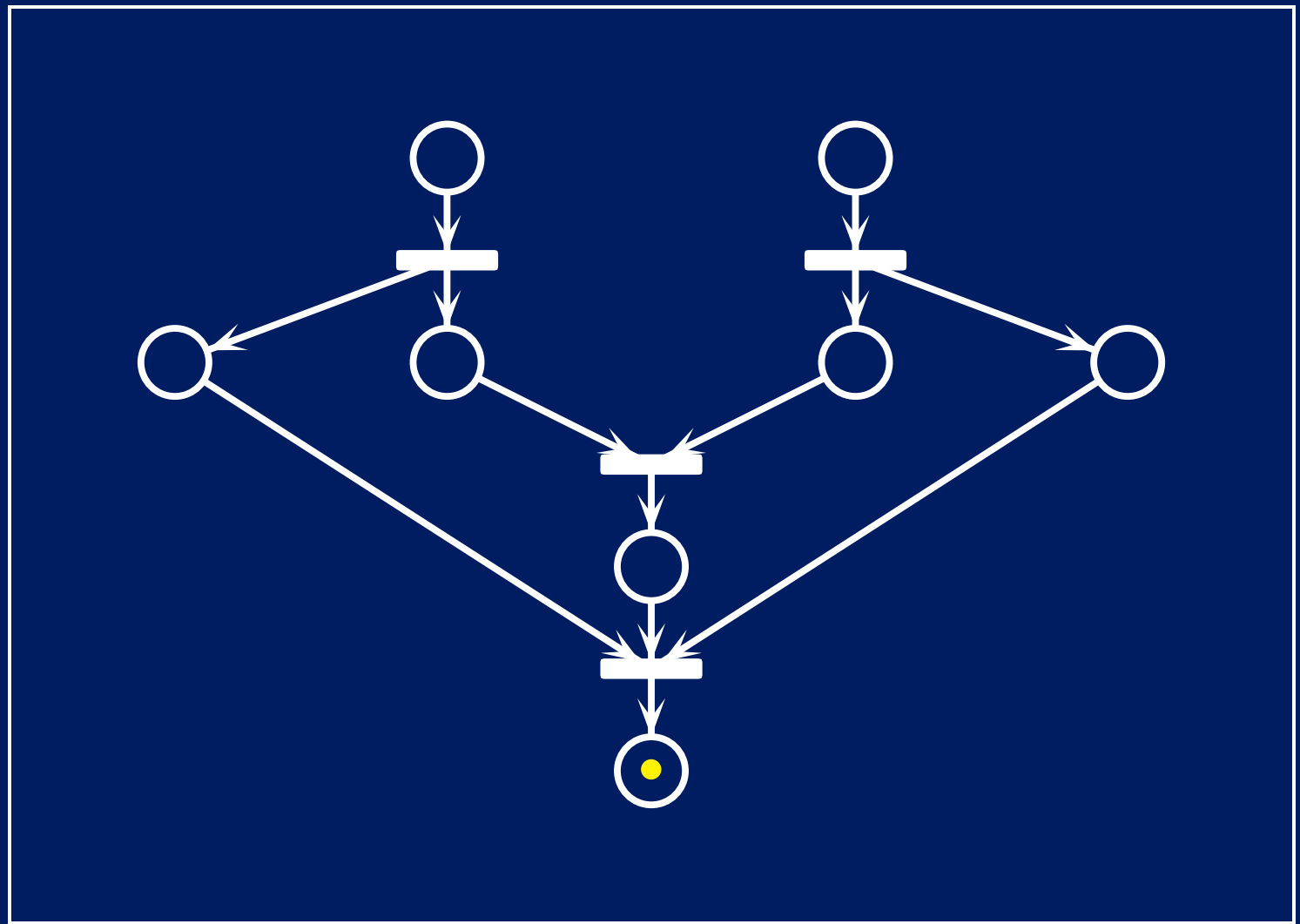

Figure 10: A more complicated Petri net.

Theoretical solution

Modelling parallel processes

Key observation:

- A graph is uniquely defined by a list of nodes and their edges.

This means that you don't need to focus on the big picture.

- Each process can be modelled individually.
- Processes are linked based on the in- and outputs.

Practical solution

make

"Utility that automatically builds programs and libraries from source code by reading files called makefiles which specify how to derive the target program."

With a couple of minor alterations we can also use it for pipelines.

- Source code $\Rightarrow$ Raw data
- Program $\Rightarrow$ Result
- Building $\Rightarrow$ Analysing

http://en.wikipedia.org/wiki/Make_(software)

Practical solution

The anatomy of a Makefile

```
1 target: prerequisites
2   recepe
```

Listing 10: Makefile snippet.

In the recipe, some special variables are available:

$@	Name of the target.
$<	The first prerequisite.
$^	All prerequisites.

Suffix rules

The % can be used for implicit targets, e.g., %.bam: %.sam

Practical solution

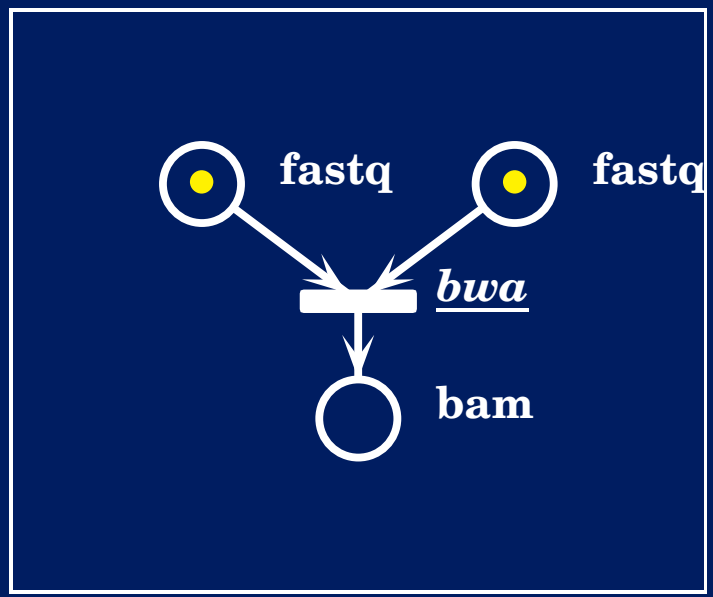

Figure 11: A simple workflow.

```
1 %.bam: %_1.fq %_2.fq
2    bwa sampe reference.fa $^ > $@
```

Listing 11: Makefile snippet.

Practical solution

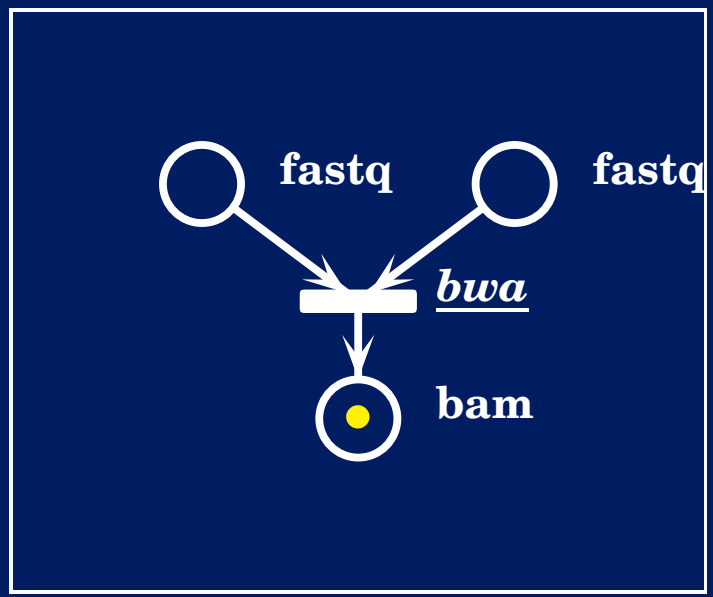

Figure 11: A simple workflow.

```
1 %.bam: %_1.fq %_2.fq
2    bwa sampe reference.fa $^ > $@
```

Listing 11: Makefile snippet.

Practical solution

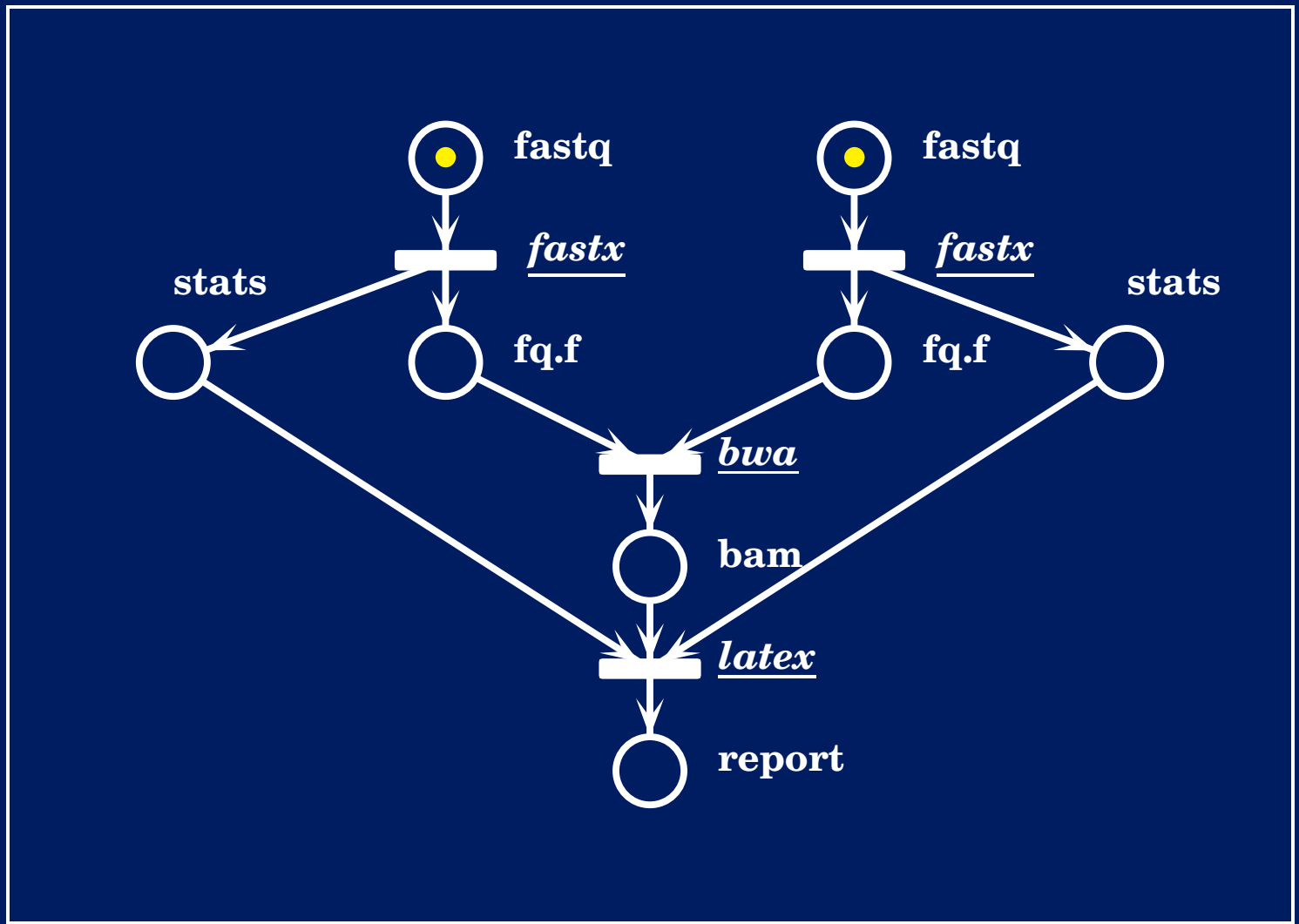

Figure 12: A parallel workflow.

Practical solution

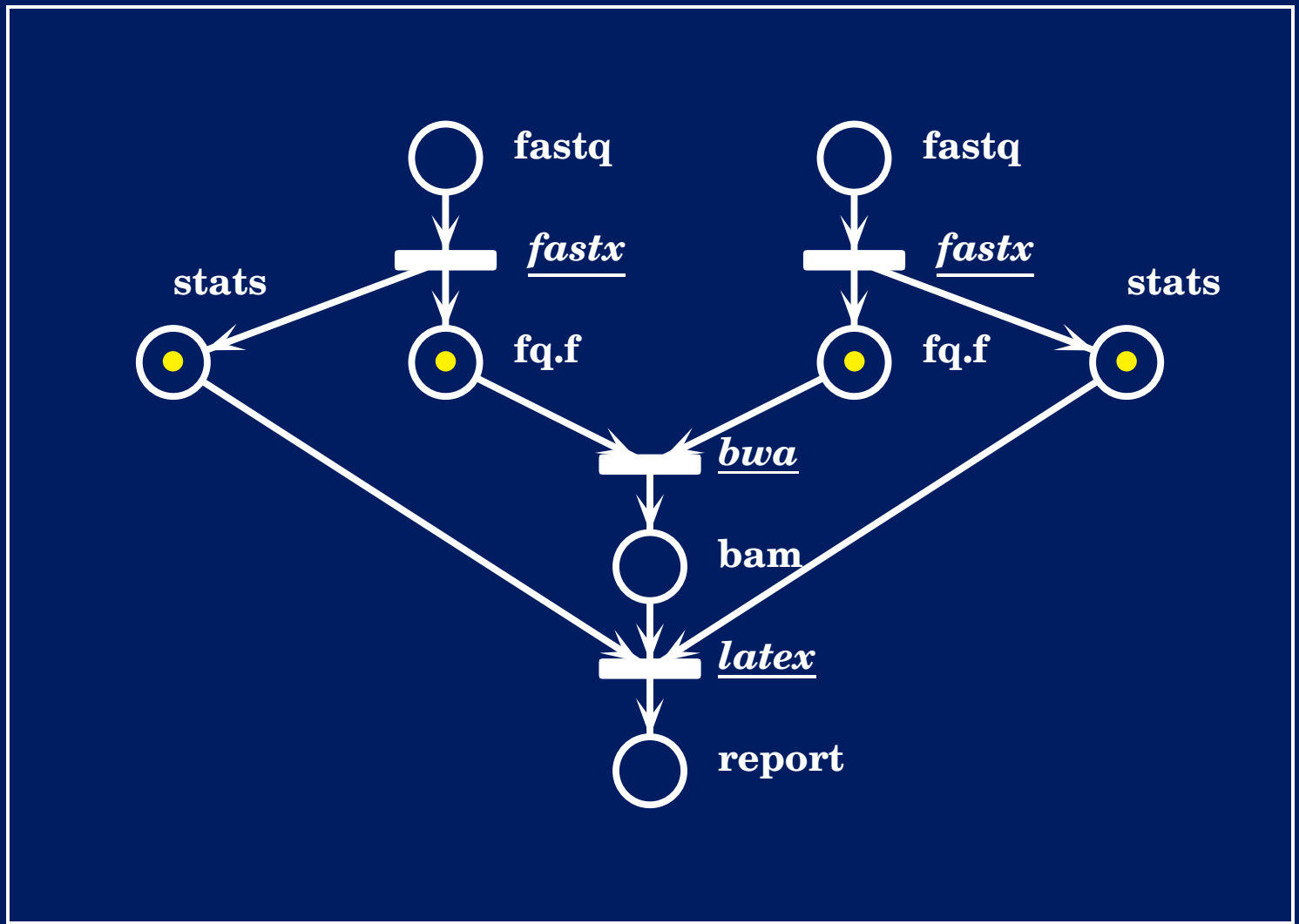

Figure 12: A parallel workflow.

Practical solution

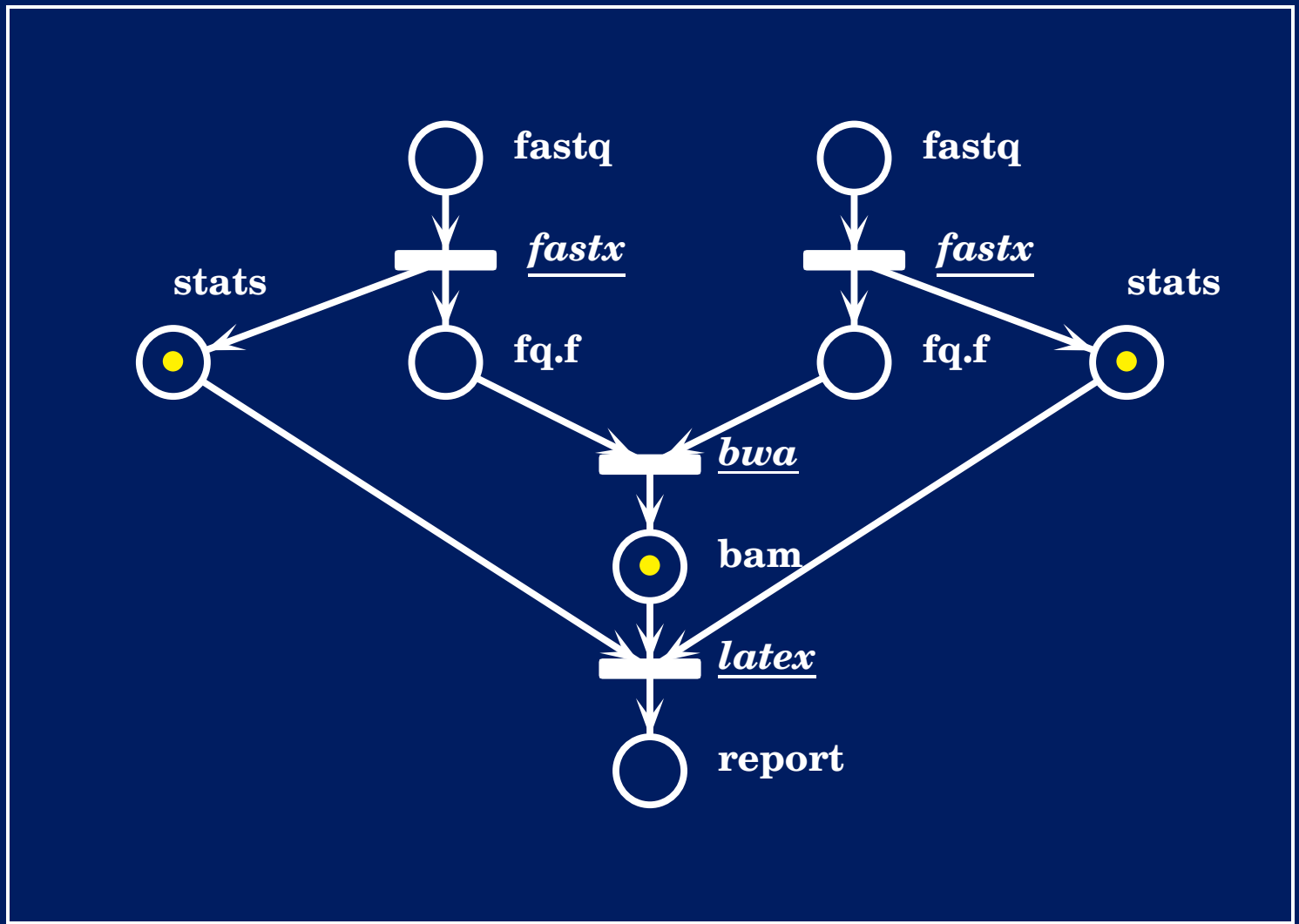

Figure 12: A parallel workflow.

Practical solution

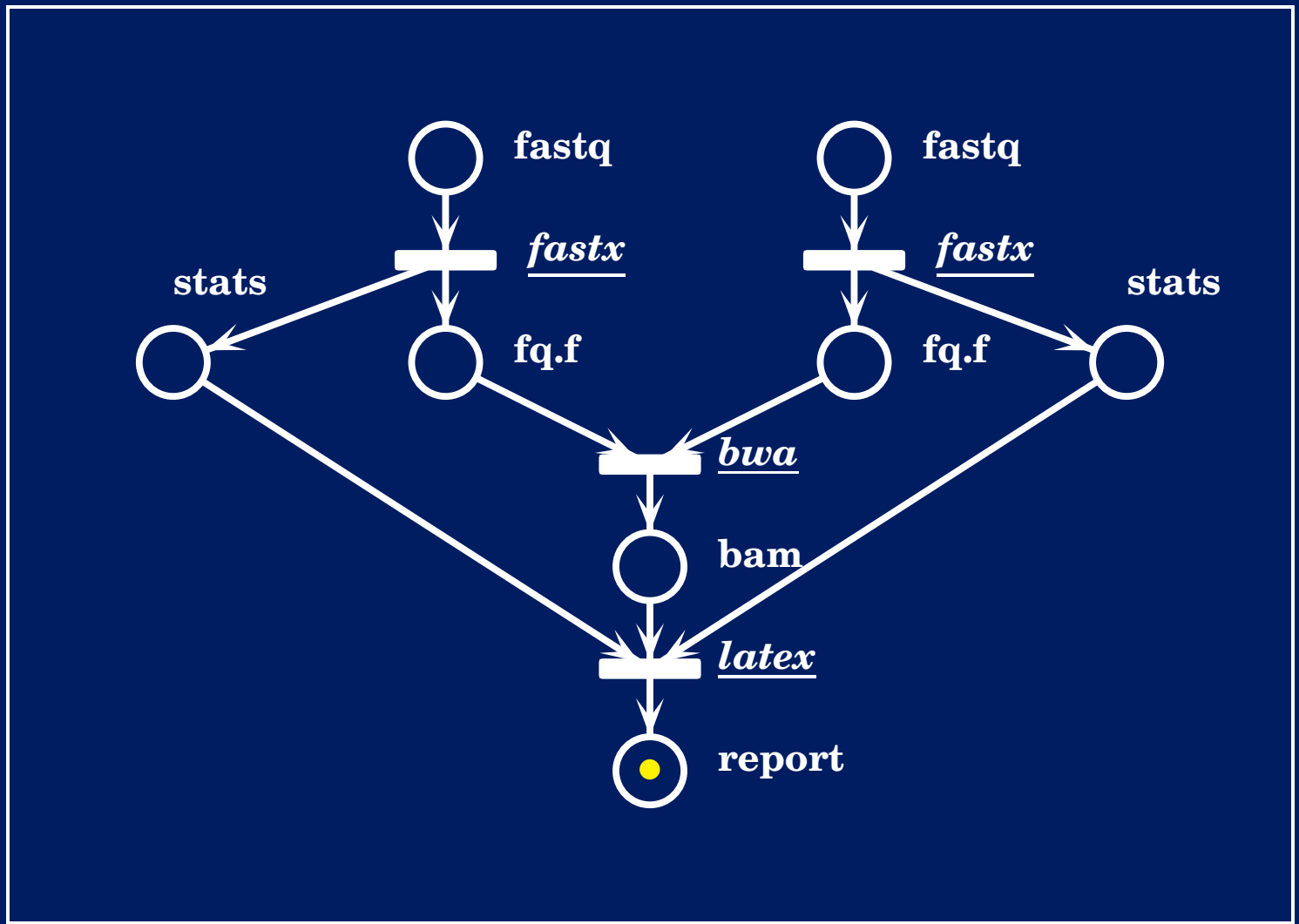

Figure 12: A parallel workflow.

Practical solution

Advantages of make

Control flow:

- Implicit.
- The target is removed if the recipe returns an error.
- The build process can resume from any point.
- Temporary files can automatically be removed.

Practical solution

Advantages of make

Control flow:

- Implicit.
- The target is removed if the recipe returns an error.
- The build process can resume from any point.
- Temporary files can automatically be removed.

Plus:

- Portable.
  - `make`, `qmake`, `nmake`, ...
- Modular.
  - Multiple (overlapping) pipelines can be combined.
- Test without executing (`make -n`).

Practical solution

Example Makefile

Use variables for the reference and binaries.

```
1 REFERENCE := /Genomes/H.Sapiens/hg19/reference.fa
2
3 BWA       := /usr/local/bwa-0.6.2/bwa
4 SAMTOOLS  := /usr/local/samtools-0.1.18/samtools
```

Listing 12: Makefile snippet.

Practical solution

Example Makefile

Use variables for the reference and binaries.

```
1 REFERENCE := /Genomes/H.Sapiens/hg19/reference.fa
2
3 BWA       := /usr/local/bwa-0.6.2/bwa
4 SAMTOOLS  := /usr/local/samtools-0.1.18/samtools
```

Listing 12: Makefile snippet.

Because:

- Full paths force using the same version.
- Changing to to a new version is easy.

Practical solution

Example Makefile (2)

This is the same pipeline as shown in Listing 2.

```
1  %.sai: %.fq
2      $(BWA) aln $(REFERENCE) $< > $@
3
4  %.sam: %_1.sai %_2.sai %_1.fq %_2.fq
5      $(BWA) sampe $(REFERENCE) $^ > $@
6
7  %.bam: %.sam
8      $(SAMTOOLS) view -bt $(REFERENCE) -o $@ $<
9
10 %.sorted.bam: %.bam
11     $(SAMTOOLS) sort $< $*.sorted
12
13 %.mpileup: %.sorted.bam
14     $(SAMTOOLS) mpileup -f $(REFERENCE) $< > $@
```

Listing 13: Makefile snippet.

GAPSS3

Resequencing analysis pipeline

First version:

- Frame written in **`bash`**.
- Custom scripts for job handling.
- Complicated to redeploy interrupted analysis.
- Modularity is limited.

https://humgenprojects.lumc.nl/trac/GAPSS3

GAPSS3

Resequencing analysis pipeline

First version:

- Frame written in **`bash`**.
- Custom scripts for job handling.
- Complicated to redeploy interrupted analysis.
- Modularity is limited.

Current version:

- Reduced code from ±2000 to ±400 lines.
- Optimised cluster usage.
- Shorter analysis time.

https://humgenprojects.lumc.nl/trac/GAPSS3

GAPSS3

The GAPSS3 workflow

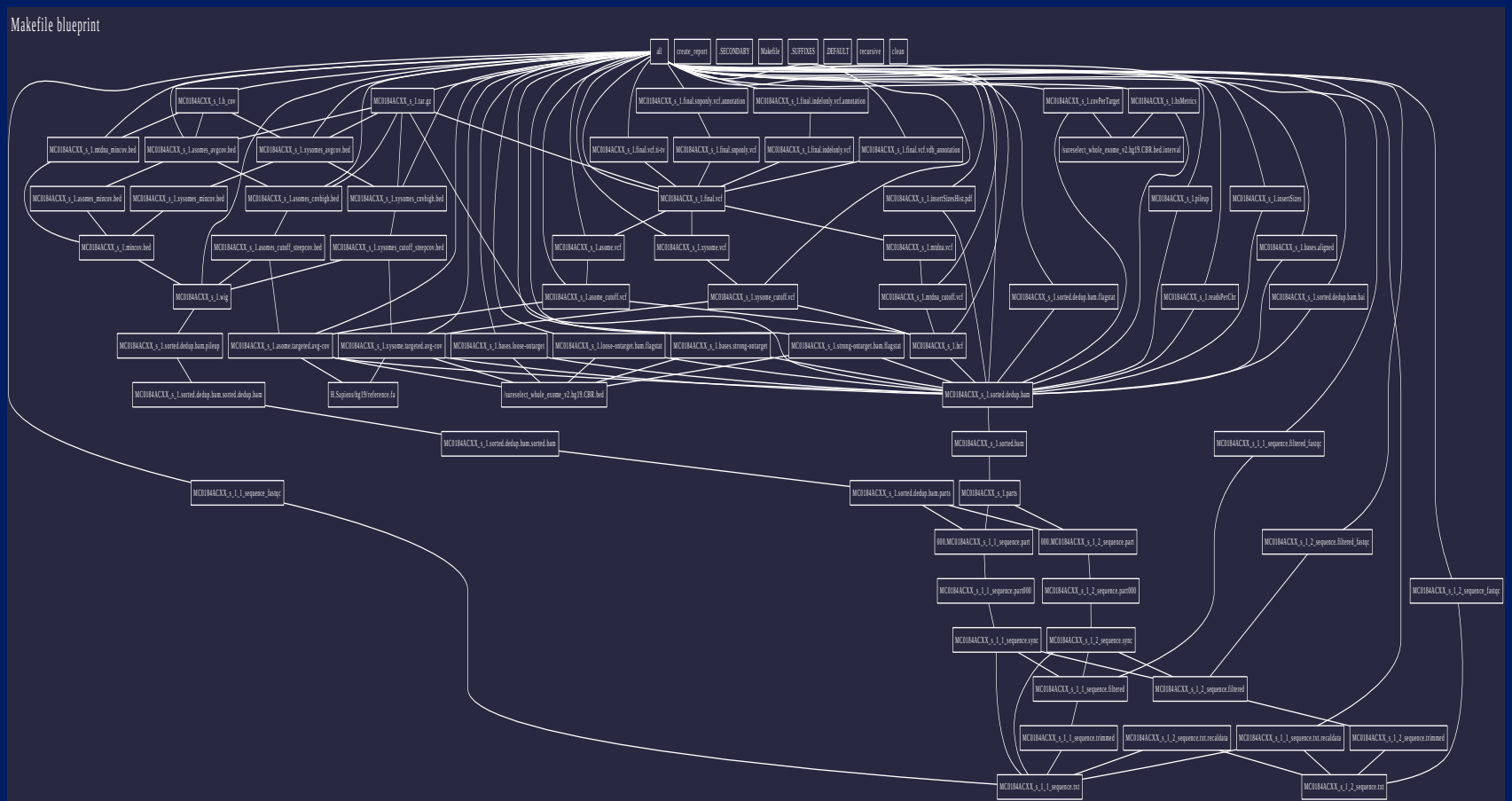

Figure 13: GAPSS3.

GAPSS3

The GAPSS3 workflow

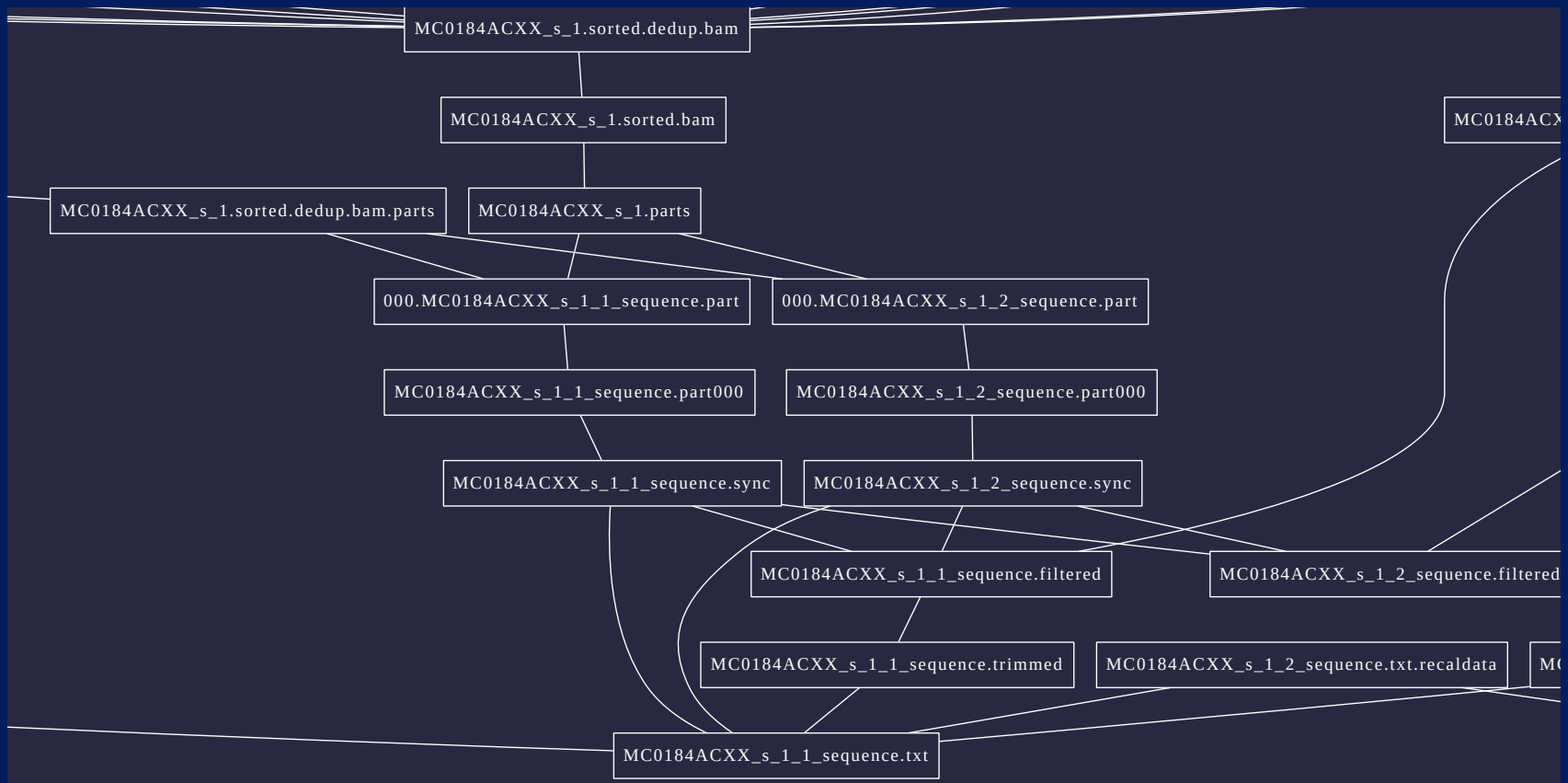

Figure 14: Part of GAPSS3.

GAPSS3

Resequencing

Exome:

- Only look at the genes.
- Will not detect everything.

GAPSS3

Resequencing

Exome:

- Only look at the genes.
- Will not detect everything.

Full genome:

- Analyse everything.

GAPSS3

Resequencing

Exome:

- Only look at the genes.
- Will not detect everything.

Full genome:

- Analyse everything.

type	desktop	cluster
exome	4 days	5 hours
genome	one year	3 days

Table 4: Gain of using a cluster.

Questions?

Acknowledgements:

Michiel van Galen
Michel Villerius
Martijn Vermaat
Johan den Dunnen