BioMake

Chris Mungall

Berkeley Drosophila Genome Project

cjm@fruitfly.org

build networks

- Many bioinformaticians spend large amounts of time coding and running *build/make networks*
- A build network is a recipe describing the execution of a collection of *interdependent heterogeneous tasks*
  - sequence analysis pipelines
  - data ‘compilation’
    - importing, transforming and exporting data
  - LIMS
- Error prone, tedious repetitive code and hard to configure

Existing approaches

- Run tasks by hand, or with ad-hoc scripts
  - doesn't scale, leads to insanity
- Unix/GNU makefiles
  - ++: concise, generic, high level abstraction
  - --: limited expressive power, hacky
- makefile replacements (cons,scons,ant,build…)
  - geared towards software development
- Bio compute pipeline software (biopipe, enspipe)
  - excellent for certain tasks
  - not completely generic

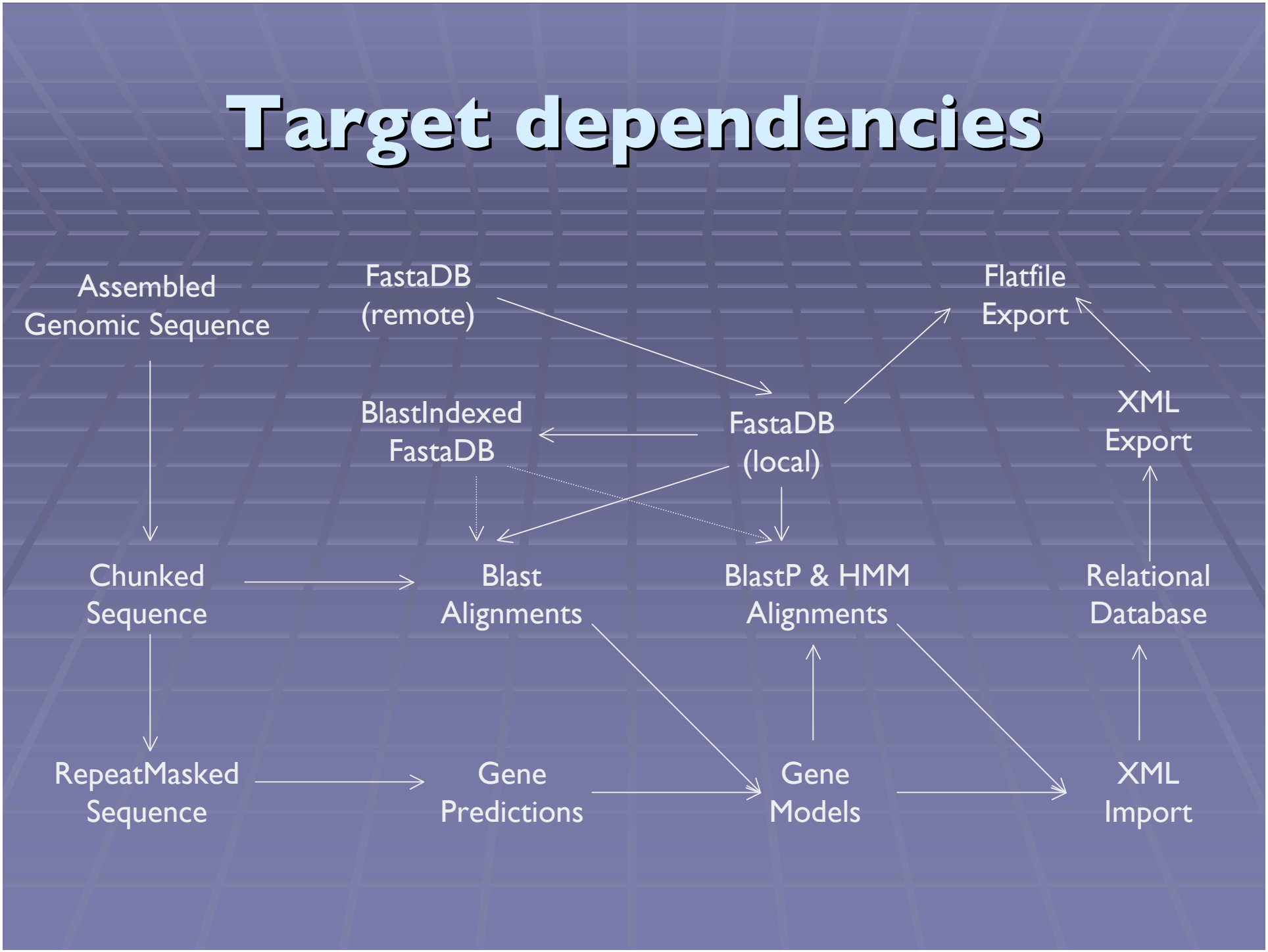
biomake: executable computational protocols

- A declarative language for specifying build networks
  - concise, Turing-complete, highly configurable
- Dependency management
  - e.g mask genomic sequence prior to genefinding
- Local and remote job execution
  - compute farm job management
- Filesystem or database oriented

Example Genomic Sequence Analysis Pipeline

- Prepare and cut assembled sequence into slices
- Download latest NR peptide dataset and index it
- Blast genomic slices against NR and other datasets
- RepeatMask genomic slices and run genefinders on masked sequence
- Synthesise gene models and do peptide analysis on results
- Store everything in a relational db, and prepare files for export to public

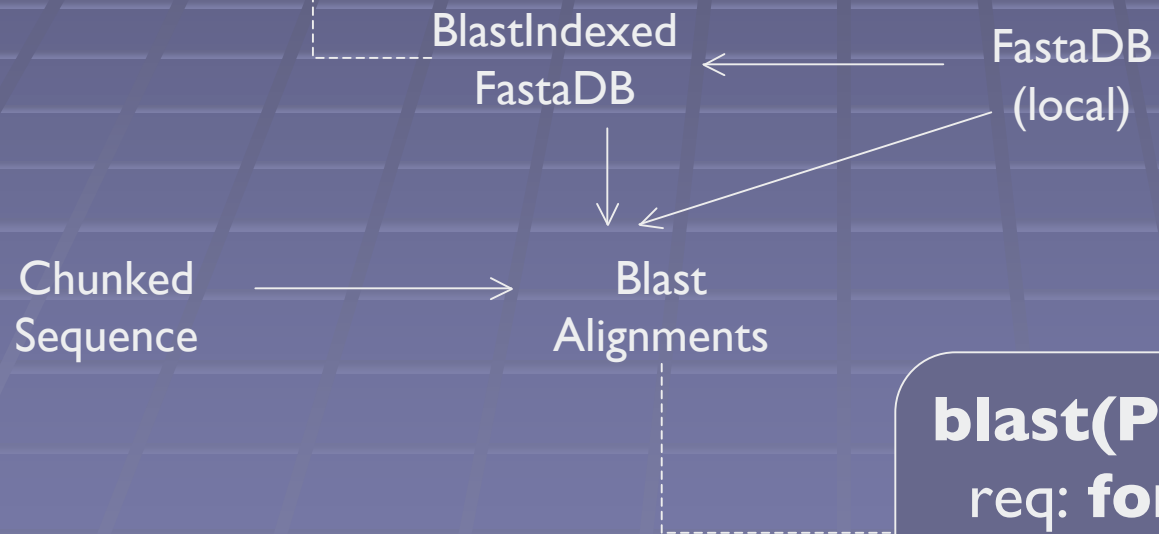
Target dependencies



Specifying Targets

A generic target pattern has a name and arguments

formatdb(D)
run: formatdb -i **D**



blast(P,S,D,A)
req: **formatdb(D)**
run: blastall -p **P** -i **S** -d **D** **A**
flat: S-blast/**bn(S).D.P**.out

A target pattern has tags for specifying dependencies, actions and filesystem or database IDs

BioMake Execution

TARGET:
blast(blastx, my.seq, nr.fly, -)

SUBTARGET:
formatdb(nr.fly)

RUN: formatdb -i nr -p T
OUT: nr.fly.{psq,pin,phr}
formatdb-nr.fly.OK

RUN:
blastall -p blastx -i my.seq -d nr.fly

OUT:
my.seq-blast/my.seq.nr.fly.blastx.out
my.seq-blast/my.seq.nr.fly.blastx.out.OK

blast(P,S,D,A)
req: **formatdb(D)**
run: blastall -p **P** -i **S** -d **D** **A**
flat: S-blast/**bn(S).D.P**.out

formatdb(D)
run: formatdb -i **D**

Targets can be nested

store(bop(my.seq, genscan(repeatmask(my.seq, drosophila))))

target instantiations can be thought of as a *skolem IDs*

repeatmask(S,Org)
run: repeatmask **S** -a **Org**
flat: **S**.mask

genscan(S)
run: genscan **S**
flat: **S**-pred/**bn(S)**.genscan.out

bop(S,B)
run: apollo -bop -s **S B** -o **target**
flat: **B**.game.xml

store(XML)
run: xml2db **XML**

Iteration

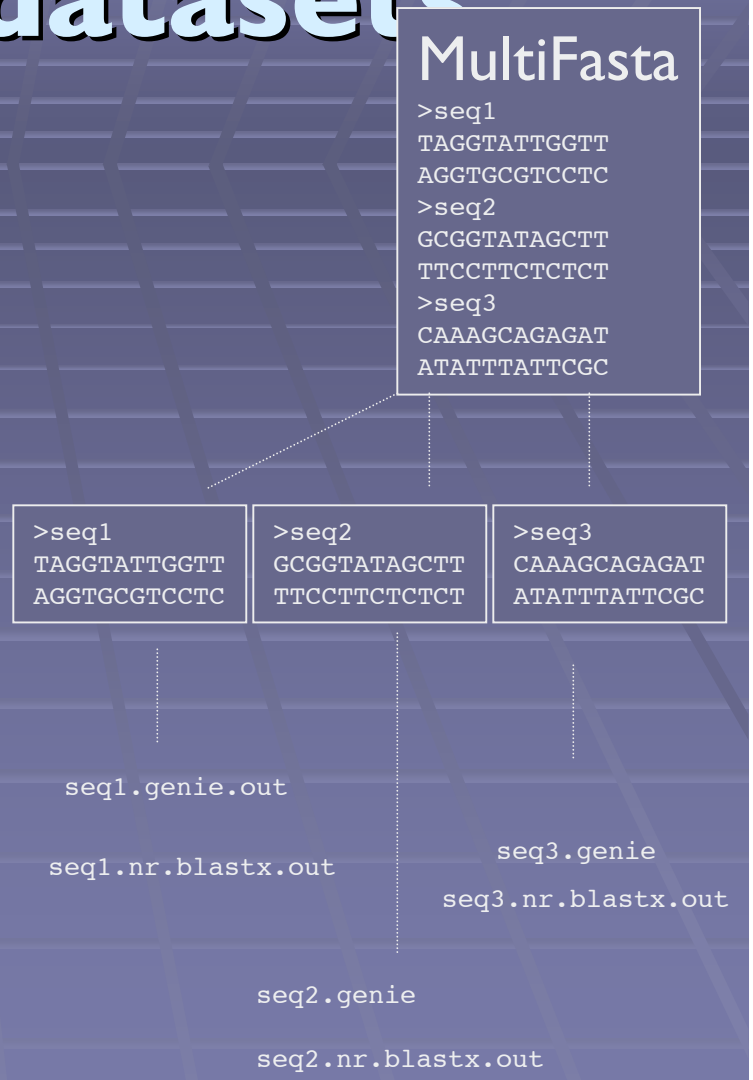
- Pipelines frequently involve iterating over collections of data:
  - Perform a sequence analysis on every entry in a multi-fasta format file
  - Perform a peptide analysis on every gene prediction in some genscan output
  - Query a database for a list of IDs and perform some task on each
- biomake has language constructs for iteration

Iterating over datasets

analyze_multifasta(F)
iterate: **analyze_seq(S)**
where **S** in **splitfasta(F)**

splitfasta(F)
run: splitfasta.pl -d **F**-split -md5 **F**
flat: **F**-split/**bn(F).**pathlist
comment: *splitfasta.pl is part of the biomake distro*

analyze_seq(S)
req: **genie(S)**
blast(blastx**,S,**nr**,-)**



Controlling the runmode

- Tasks can be run locally or on a compute farm, synchronously or asynchronously
  - wrapper provided for PBS
- *runmode:* tag states the mode and wrapper for a particular target pattern
  - can be set globally and per-pattern
- special status targets provide execution status

runmode example

blast(P,S,D,A)
req: **formatdb(D)**
run: blastall -p **P** -i **S** -d **D** **A**
flat: S-blast/**bn(S).D.P**.out
runmode: async(qsubwrap)

The blast job will be executed on the compute farm via qsubwrap (comes with biomake distro)

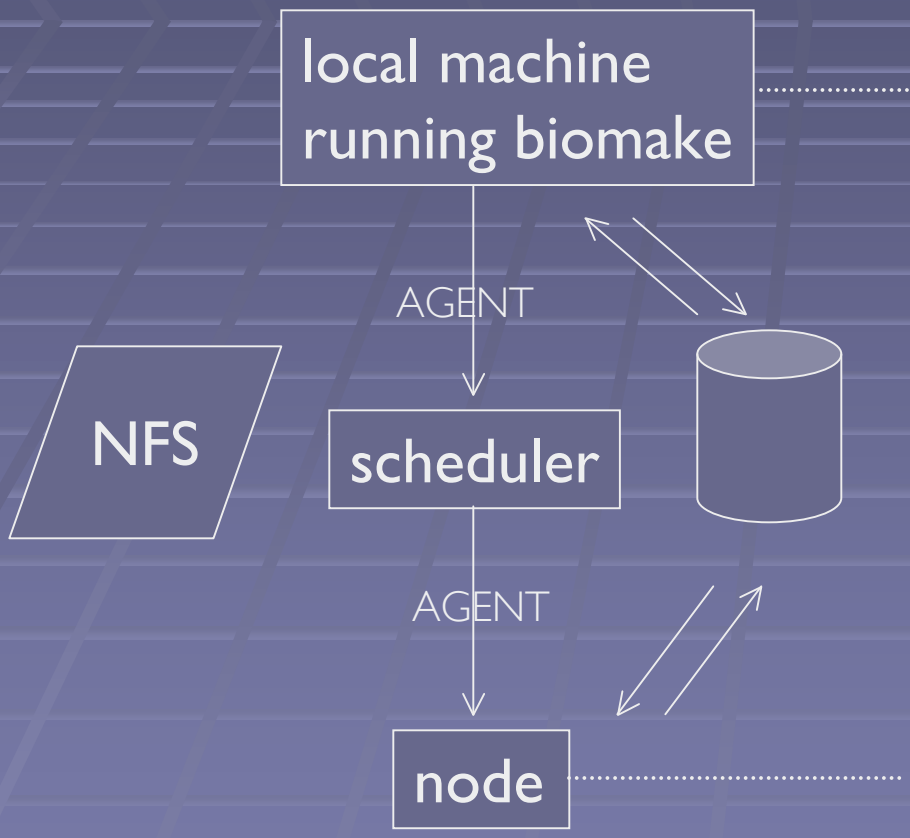
biomake can automatically handle moving data in and out between user's filesystem (or db) and local cluster nodes

Upon submission, the status target **status_run(blast(P,S,D,A))** will be generated; on completion, the target **status_ok (blast(P,S,D,A))** will be generated

Datastores

- BioMake persists targets in Datastores
- The *flat:* tag flattens targets to unique datastore IDs
- Datastore can be filesystem or relational database
  - default is filesystem
  - can be set globally or per target
    - e.g. analysis result targets can be stored on filesystem, status targets stored in DB
- NFS traffic can be avoided on compute farm by storing targets and status targets in a database

Asynchronous Execution



For each target T to be built:

1) biomake fetches status of T
   skips T if status = ok/run
2) biomake stores **status_run(T)**
3) biomake creates a runner agent script and submits it to the cluster
4) continue onto next target

1) agent **fetches** any input data
2) agent **runs** command (eg blast) synchronously
3) agent **stores** result
4) agent **stores** status of T as 'ok' or 'err'

Specifying rules

- Pipeline systems often require a rule base
  - only do nuc to nuc alignments on one species or two recently diverged species
  - use sequence ontology hierarchy to decide analyses or parameters
- biomake protocols can have prolog facts and rules embedded inside them
- biomake distro comes with SO prolog db and rules for graph traversal

Embedding prolog facts

<data relation="fastadb" cols="ID,SeqAlphabet,SOType,Org" del="ws">

protein.fly.fst	aa	polypeptide	D melanogaster
est.fly.fst	na	EST	D melanogaster
cdna.fly.fst	na	cDNA	D melanogaster

</data>
<prolog>
nucdb(D):- fastadb(D,na,_,_).
pepdb(D):- fastadb(D,aa,_,_).
</prolog>

formatdb(D)
run: formatdb -i D -p 'T' {pepdb(D)}
run: formatdb -i D -p 'F' {nucdb(D)}

BioMake module system

- The biomake core language is generic
  - no bioinformatics-specific code or tweaks
- biomake uses a module system
- biomake distro comes with
  - biosequence_analysis module
  - Sequence Ontology prolog db and rules
  - scripts for handling bioinformatics data

biomake extensibility

- biomake is a declarative language
- embodies both logical and functional paradigms
- targets are actually higher order functions
- standard FP functions available in fp module
  - cons,map,grep,filter,fold,…
- goal: expressive power, concise specifications and simplicity

BioMake in use

- currently we're using biomake for…
  - analysis of repeat families found in orthologous and paralogous introns
  - Building the Gene Ontology database
- …but we are still dependent on legacy pipeline code for many analyses

Running biomake

- Get distro from http://skam.sourceforge.net
- Requires XSB Prolog
  - http://xsb.sourceforge.net
- Run via command line
  - similar to unix make command
- Works on both OS X and Linux
- Relational datastore requires mysql (Pg soon)
- Better docs coming soon, lots of examples

Acknowledgements

Shengqiang Shu
Sima Misra
Erwin Frise
Eric Smith
Mark Yandell
George Hartzell
Chris Smith
Simon Prochnik
Jon Tupy
Josh Kaminker
Karen Eilbeck
Nomi Harris
Suzanna Lewis
Gerry Rubin

Problem Specification

- A build network consist of multiple *Targets*
  - e.g the output from a **blastx** alignment of **my.seq** to the protein database **nr.fly**
- Targets have a *logical pattern*
  - e.g **blast** alignment using **P** of some seq **S** vs some db **D**
- Targets are *dependent* on other Targets
  - e.g blast depends on the indexing of db **D** using **formatdb**
- Upstream changes trigger downstream actions
- Targets are built by *running actions*
  - "**formatdb –p T –i nr.fly**"
  - "**blastall –p blastx –i my.seq –d nr.fly**"