Industrial Personalised Immunotherapy Pipeline Development with Nextflow

Luke Goodsell

Team Leader – Bioinformatics Software Development, Achilles Therapeutics

22nd November 2018

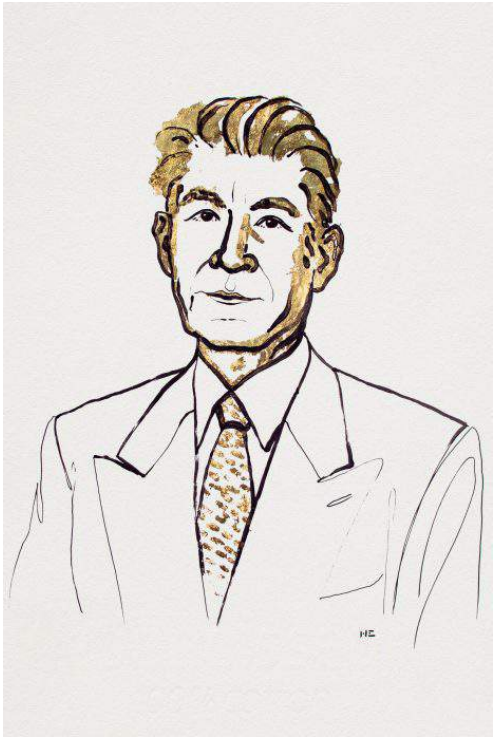
Disclaimer

- Paid employee of Achilles Therapeutics
- Views expressed are my own
- Talking today about technical implementation

Contents

- Precision Immuno-oncology
- Industrial Pipeline Development
- Implementation in Nextflow
- Continuous Integration in Nextflow

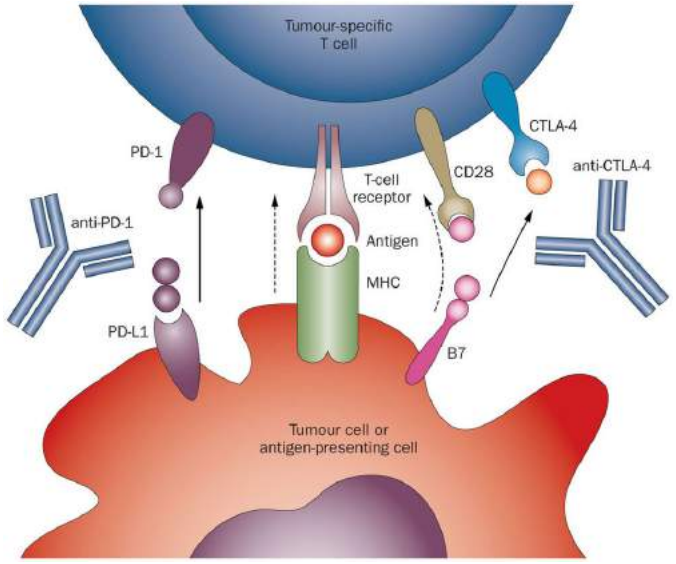
Cancer Immunotherapy



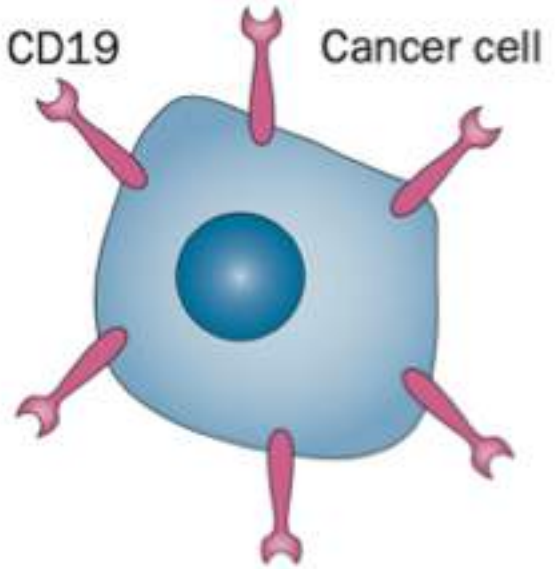
Tasuku Honjo and James Allison
The Nobel Prize in Physiology or Medicine 2018

Immune system distinguishes cancerous from normal cells through tumour antigens

- Cancer immunotherapy activates the immune system to destroy tumour cells.
- The immune system must distinguish tumour from normal cells using tumour antigens.
- These antigens could be on the cell surface due to:
  - Presentation via the major histocompatibility complex (MHC-I/II).
  - Natural extracellular markers (e.g. CD19)

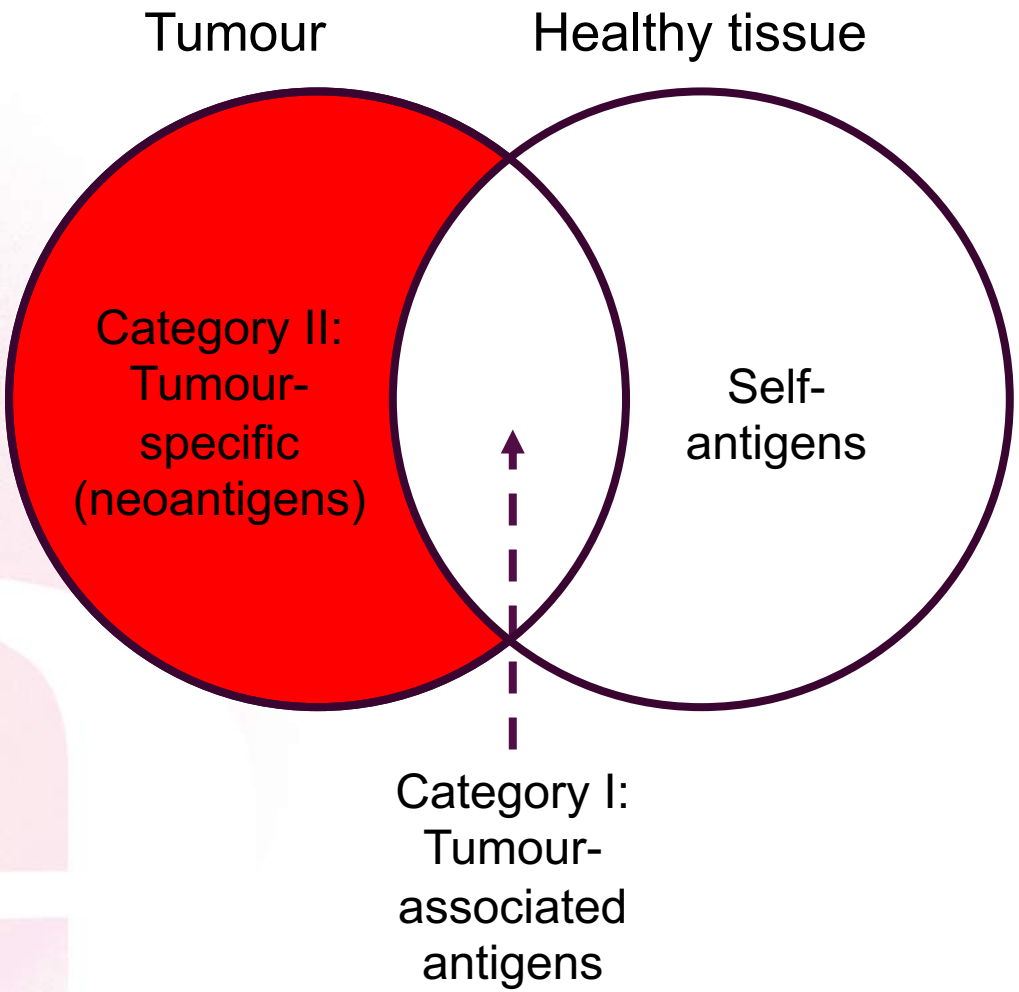


https://media.nature.com/full/nature-assets/nrclinonc/journal/v11/n1/images/nrclinonc.2013.208-f2.jpg



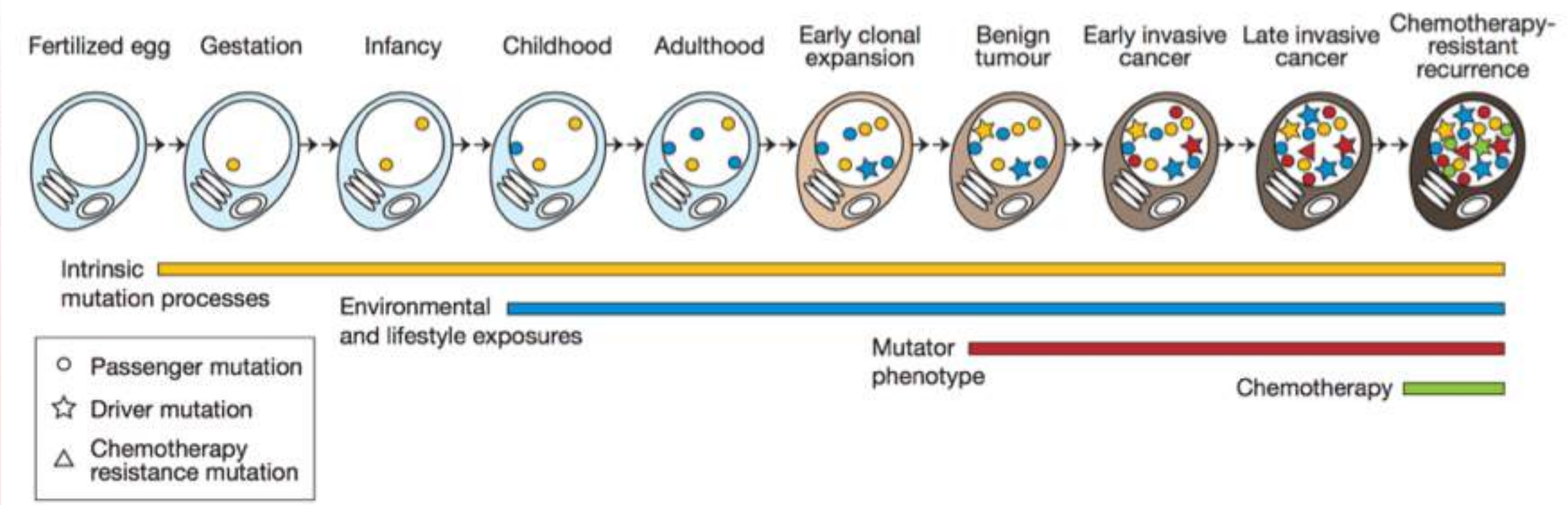
James N Kochenderfer and Steven A Rosenberg. 2013. Nature Reviews Clinical Oncology.

Tumour-specific antigens (neoantigens)



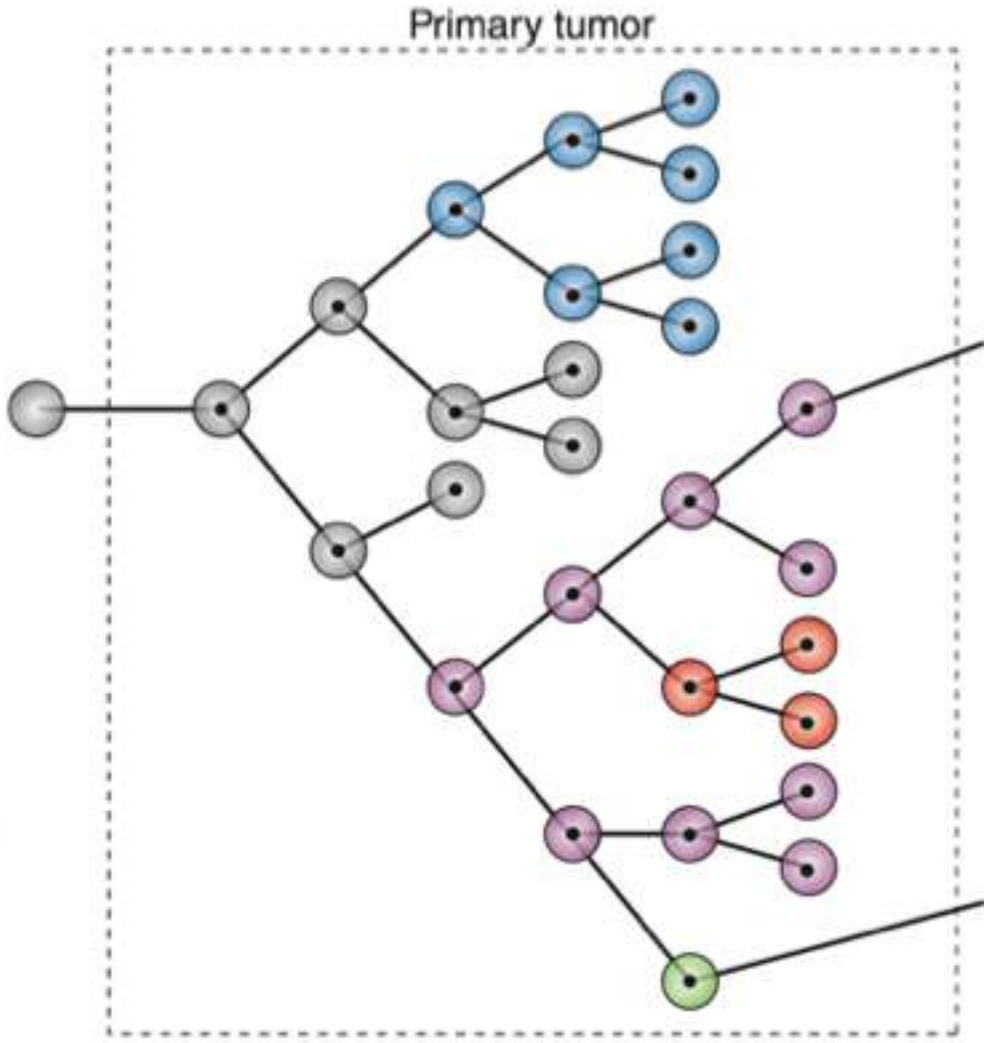
Cancer is a genetic disease

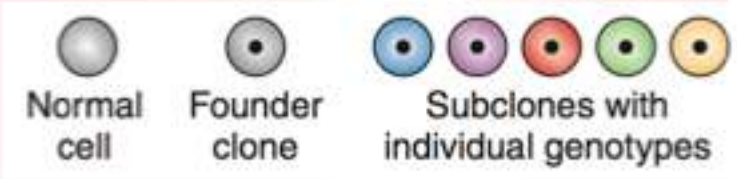
- Development of cancer through progressive accumulation of mutations over the lifespan of an individual cancer patient.



Stratton M, Campbell PJ, and Futreal A. Nature (2009)

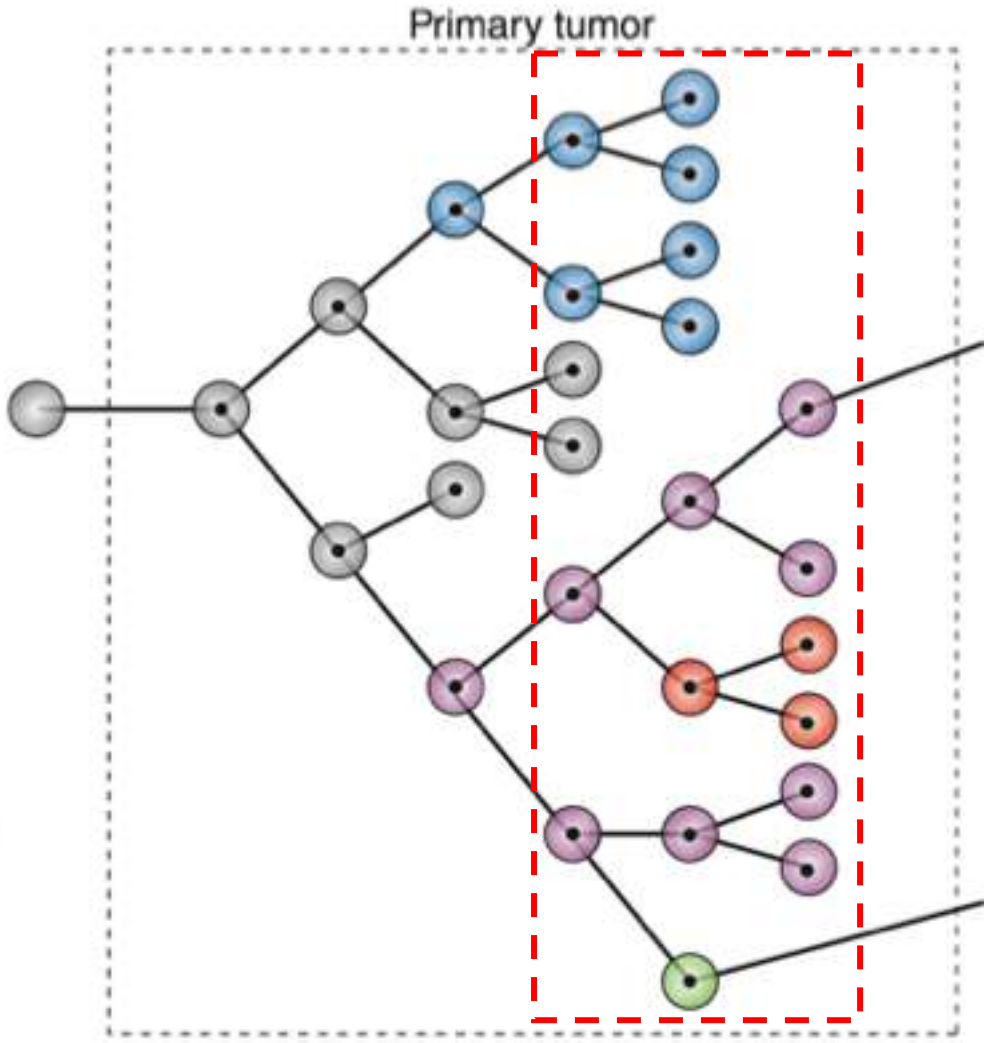
Cancer development is a branched evolutionary process – Clonal evolution

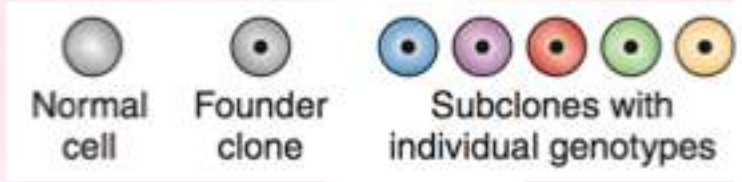




Modified version of Fig 1. Caldas C. Nature Biotechnology (2012).

Tumour heterogeneity is a consequence of clonal evolution

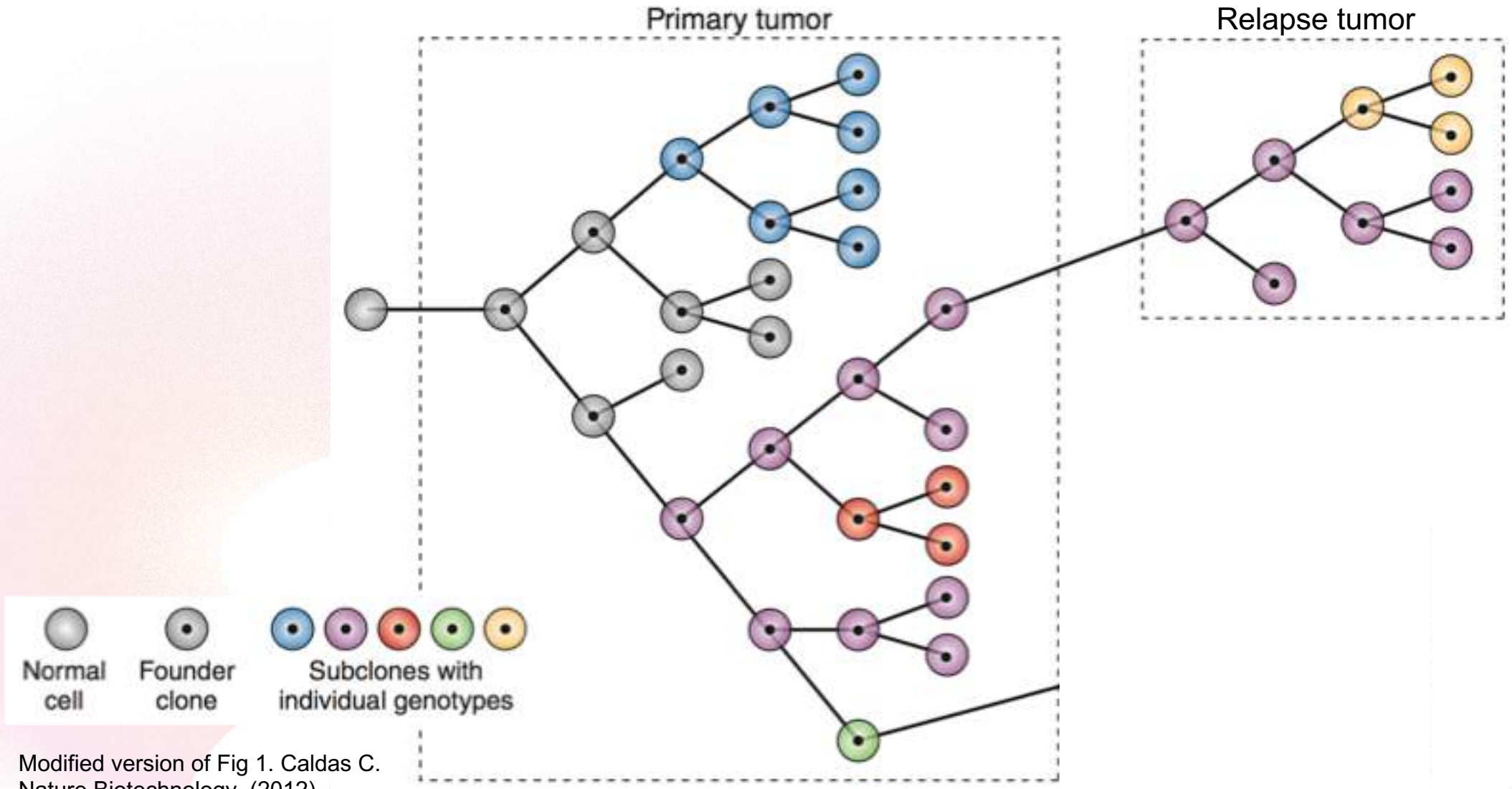




Modified version of Fig 1. Caldas C. Nature Biotechnology (2012).

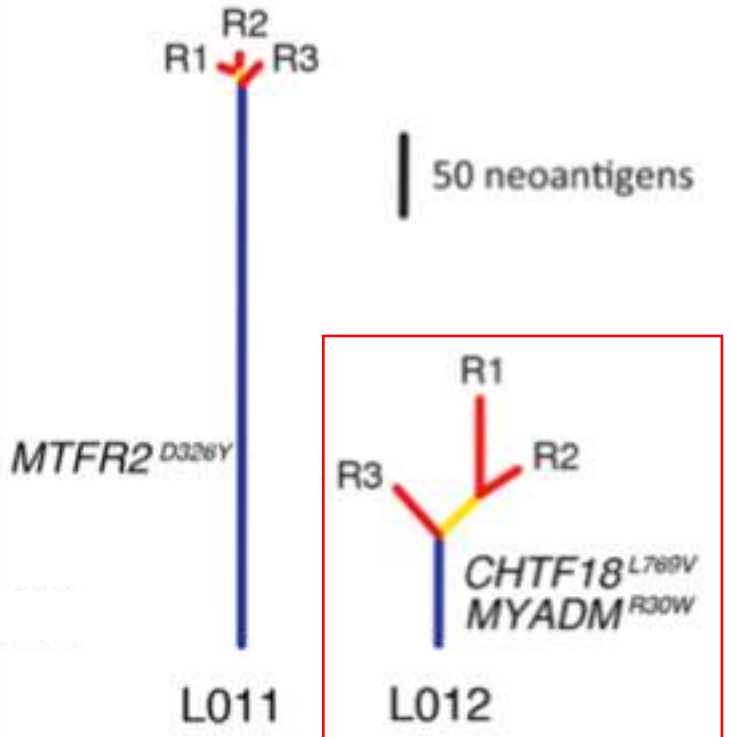
A “snapshot” of a tumour is often **represented by the profiles of multiple clones** that have undergone clonal evolution

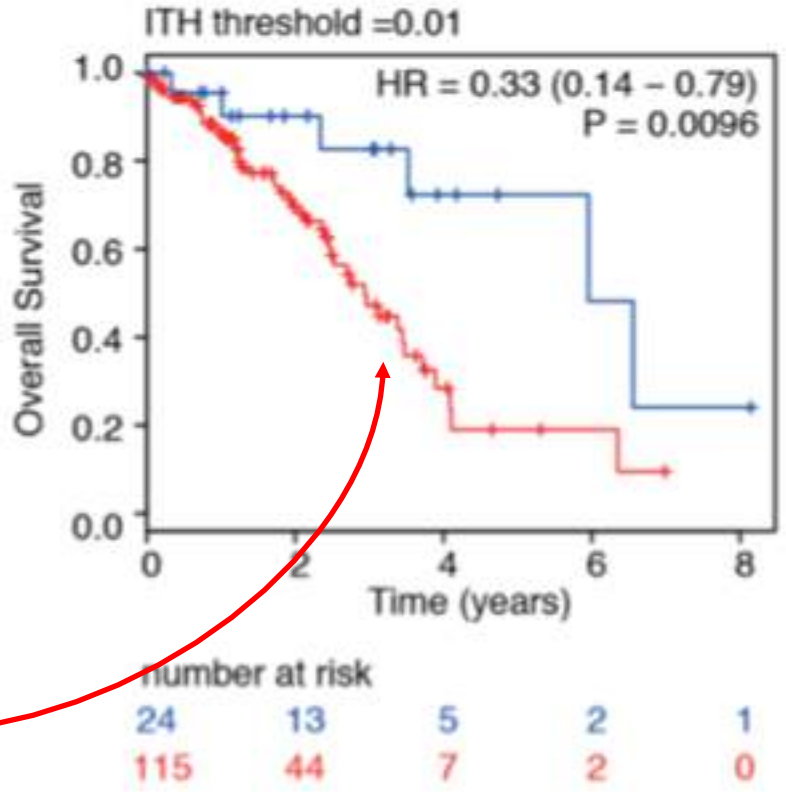
Phenotypic variation provides the substrate for natural selection



Modified version of Fig 1. Caldas C. Nature Biotechnology (2012).

Patients with clonal neoantigens have a better overall survival

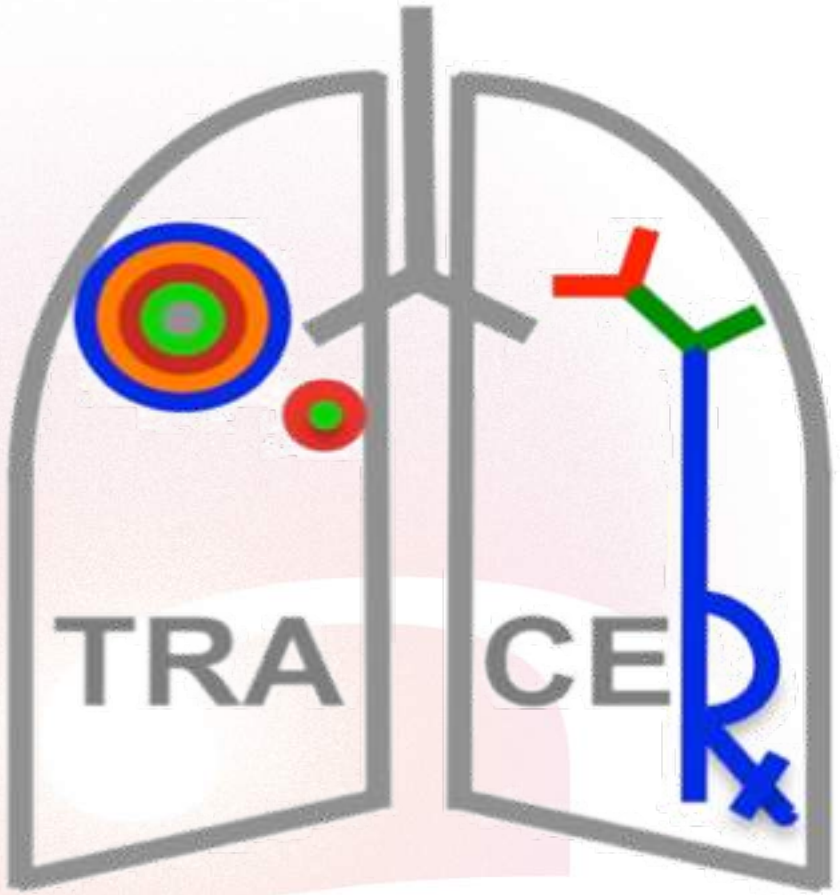




McGranahan N*, Furness AJS*, Rosenthal R* *et al*. Science. (2016)

Industrial Pipeline Development

TRACERx



- An ongoing large study of the mechanisms of cancer evolution.
- Makes use of large patient cohort, ~850 individuals.
  - Tumour resections and metastasectomies
  - High-depth (~500X) whole-exome sequencing
  - Multi region sampling
  - RNA-seq
  - Tracking of patients over time from as early as stage 1A
- Has revealed many insights into evolution of lung cancers.
  - Response to selective pressures from immune system
  - Mutational signatures
  - Driver events
- Initially lung-only, now broadened to renal, melanoma and prostate

Achilles Therapeutics

Professor Karl Peggs,
University College London

Professor Mark Lowdell,
University College London

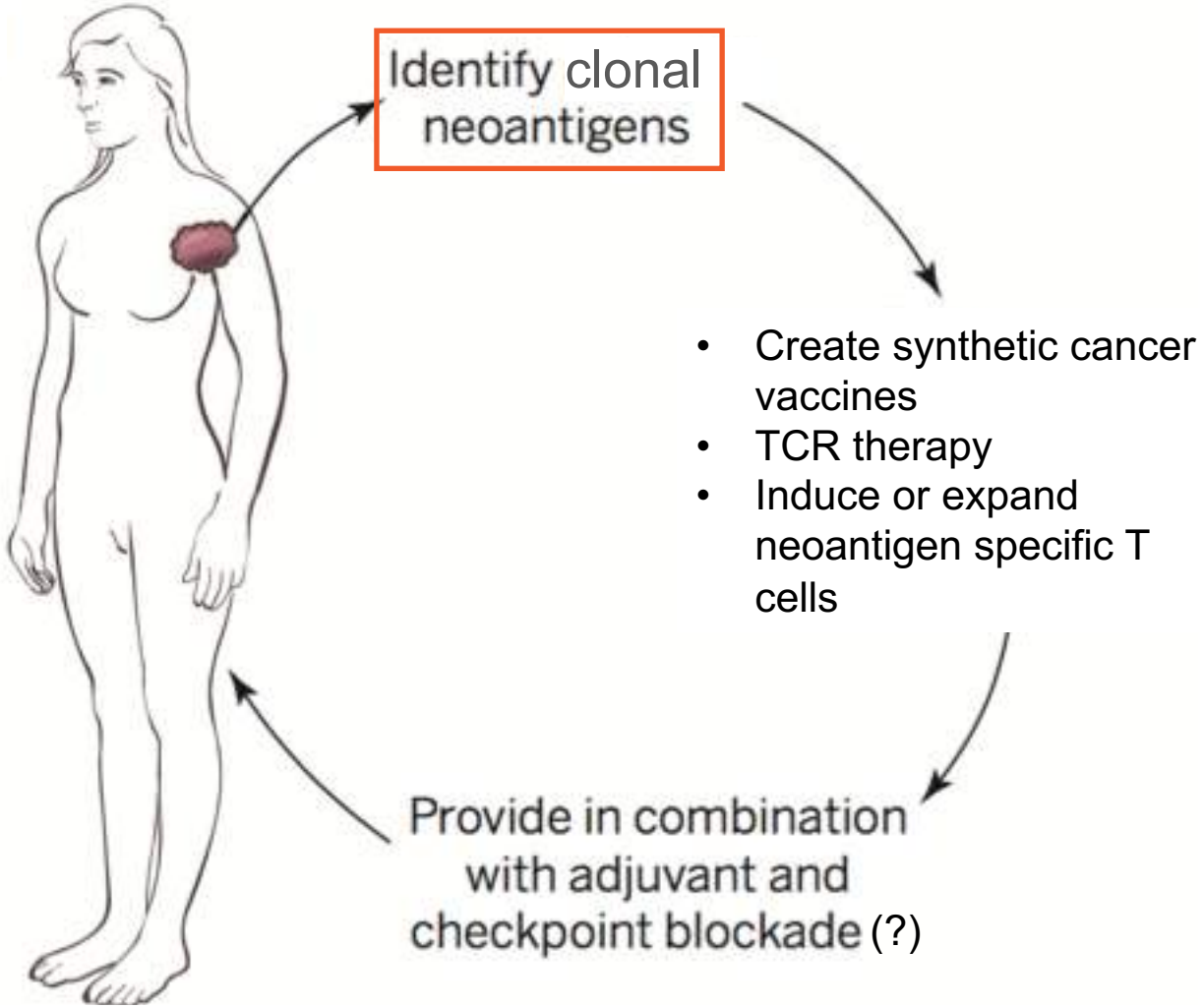
Professor Charles Swanton,
UCL & Francis Crick Institute

Professor Sergio Quezada,
University College London

Achilles at 1 year

- Founded May 2016
- Aiming to develop a large-scale precision immunotherapy targeting the clonal neoantigens
- Funded by Sycona Ltd – the venture capital arm of the Wellcome Trust

Therapy Pipeline

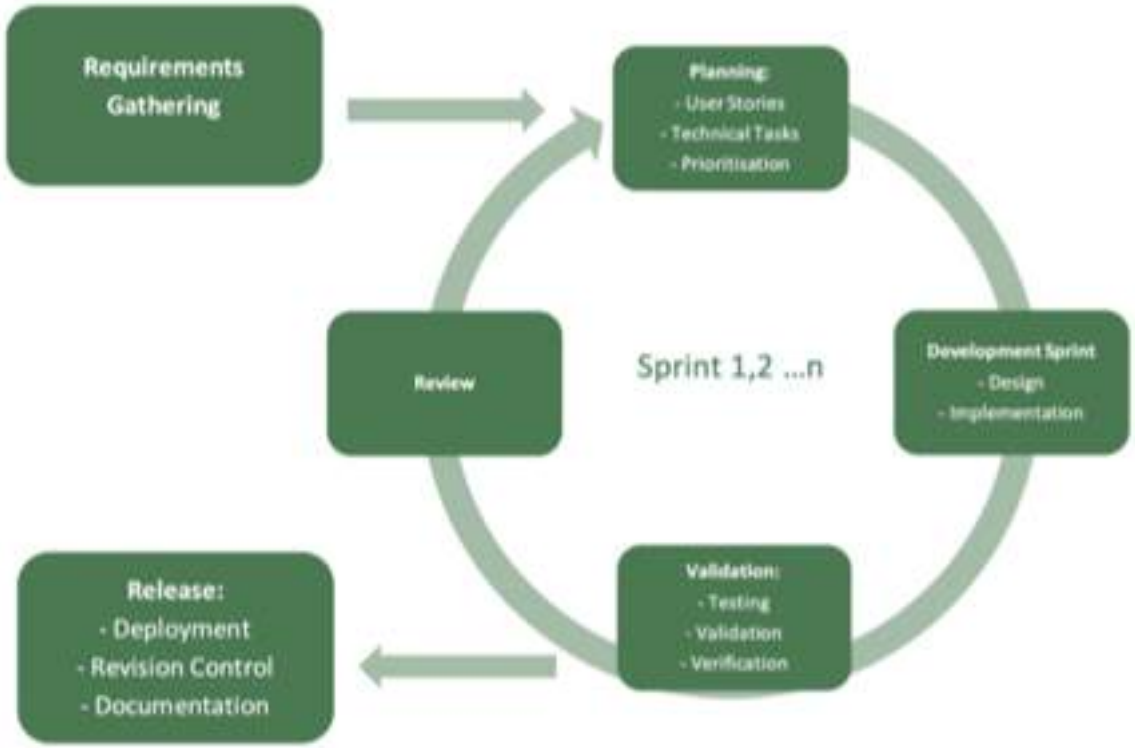


Modified from Schumacher TN. Science. (2015). Neoantigens in cancer immunotherapy.

- We are planning 2 Phase I studies one of which is in patients with locally advanced or metastatic non-small cell lung cancer and one is in another solid tumour
- These studies will start in the UK in 2019

Development Process

- The pipeline and its development need to:
  - Be flexible to changing requirements
  - Assure output quality
  - Manage patient risk
  - Ensure data security
  - Improve maintainability
  - Be fast and cost-effective
- An Agile Software Development Life Cycle, with Continuous Integration and routine validation help to achieve this

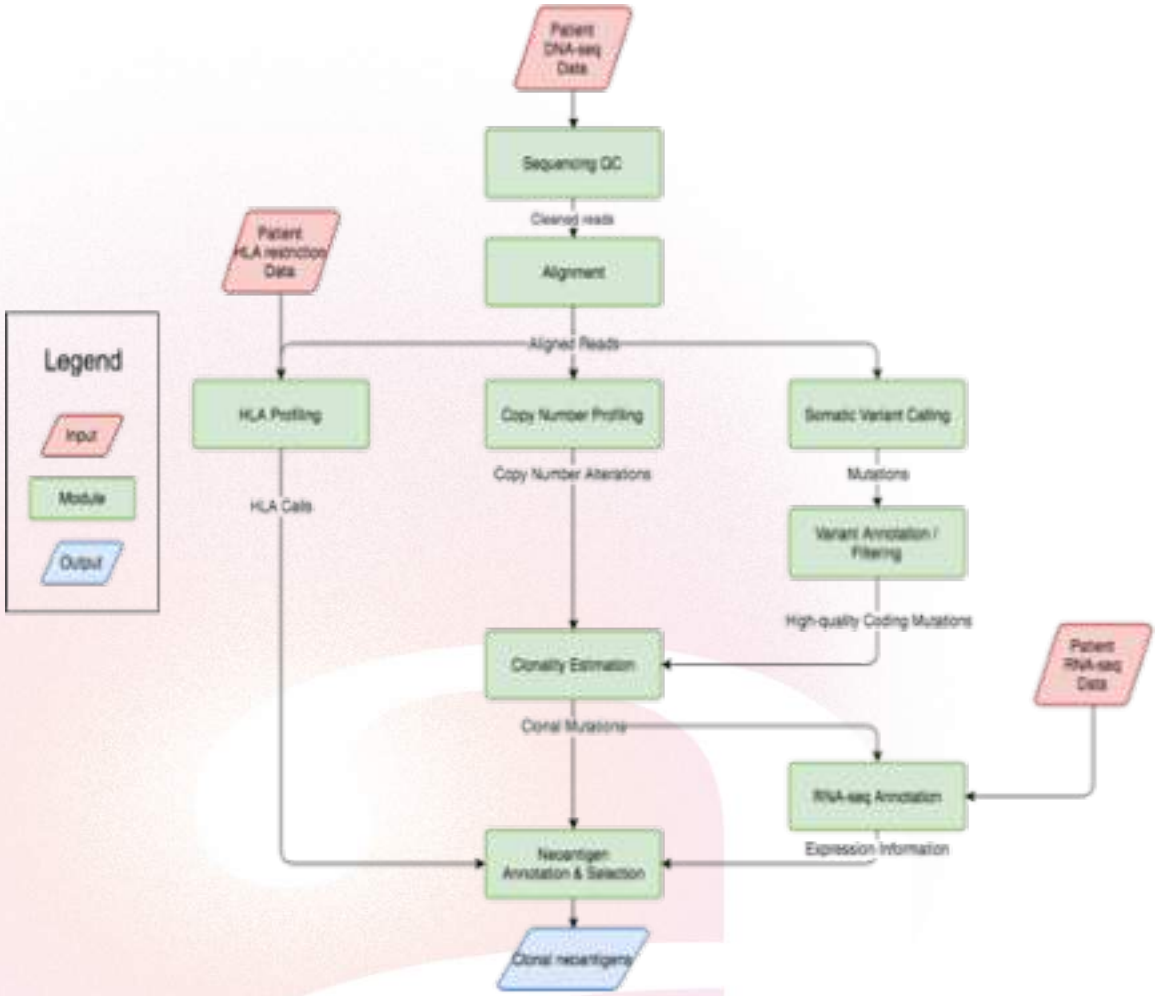


Pipeline Migration

TRACERx Pipeline	*Peleus*™ Pipeline
Research use only	Contributes to clinical process
Used for addressing many open-ended research questions	Used for addressing one specific application
Some manual intervention steps; different researchers run different portions	Runs from end-to-end without intervention by any operator
Runs on large academic infrastructure	Portable/platform-agnostic
Can use academia-only licenses	Licensed software must be justified or replaced
Patient data can wait in queue to be processed	Must be scalable to allow for surges in patient data

Peleus consigns Achilles to Chiron's care.
Credit: Wikipedia / Marsyas

Peleus™ Pipeline



- Simplified pipeline workflow
- Many discrete modules with specific outputs
- Substantial potential for with-module and between-module parallelisation.
- Different modules have different optimal hardware
- QC outputs at every step

Implementation in Nextflow

Why Nextflow?

- Inherited workflow written in R hard-coded for specific infrastructure
- Needed the pipeline to be:
  - Portable
  - Reproducible
  - Parallelisation
  - Support for many languages
  - Maintainable / future-proof
  - Fail-safe
  - Configurable
  - Scalable

Table 1 Comparison of Nextflow with other workflow management systems

Workflow	Nextflow	Galaxy	Toil	Snakemake	Bpipe
Platform[a]	**Groovy/JVM**	**Python**	**Python**	**Python**	**Groovy/JVM**
Native task support[b]	Yes (any)	No	No	Yes (BASH only)	Yes (BASH only)
Common workflow language[c]	No	Yes	Yes	No	No
Streaming processing[d]	Yes	No	No	No	No
Dynamic branch evaluation	Yes	?	Yes	Yes	Undocumented
Code sharing integration[e]	Yes	No	No	No	No
Workflow modules[f]	No	Yes	Yes	Yes	Yes
Workflow versioning[g]	Yes	Yes	No	No	No
Automatic error failover[h]	Yes	No	Yes	No	No
Graphical user interface[i]	No	Yes	No	No	No
DAG rendering[j]	Yes	Yes	Yes	Yes	Yes
Container management					
Docker support[k]	Yes	Yes	Yes	No	No
Singularity support[l]	Yes	No	No	No	No
Multi-scale containers[m]	Yes	Yes	Yes	No	No
Built-in batch schedulers[n]					
Univa Grid Engine	Yes	Yes	Yes	Partial	Yes
PBS/Torque	Yes	Yes	No	Partial	Yes
LSF	Yes	Yes	No	Partial	Yes
SLURM	Yes	Yes	Yes	Partial	No
HTCondor	Yes	Yes	No	Partial	No
Built-in distributed cluster[o]					
Apache Ignite	Yes	No	No	No	No
Apache Spark	No	No	Yes	No	No
Kubernetes	Yes	No	No	No	No
Apache Mesos	No	No	Yes	No	No
Built-in cloud[p]					
AWS (Amazon Web Services)	Yes	Yes	Yes	No	No

Di Tommaso, Chatzou, Floden, Barja, Palumbo, and Notredame. Nature Biotechnology 2017

Nextflow Modularisation

- Need to isolate modules – control file creation and access
- Clean object/class based system would be ideal (#238)
- Our solution: nested workflows
  - Workflows take parameters from YAML
  - Master workflow co-ordinates module workflows
  - Module workflows co-ordinate component workflows
  - Component workflows launch jobs remotely
- YAML easily human- or computer-generated.
  - Common chunks easily shared; process executor parameterised.

my_module_runner.nf

```
process myModuleParamsProcess {
    output:
    file("myModuleParams.yaml") into myModuleParamsChannel

    exec:
    def myModuleParams = [:]
    myModuleParams["programs"] = params.programs
    def myModuleParamsFilePath = \
        new File("${task.workDir}/myModuleParams.yaml")
    def yaml = new Yaml()
    yaml.dump(
        myModuleParams,
        new FileWriter(myModuleParamsFilePath)
    )
}

process myModuleProcess {
    storeDir "${myModuleOutputDirectory}"
    stageOutMode "move"

    input:
    file paramsFile from myModuleParamsChannel
    val myModuleWorkflow from \
        findModule("com.achillestx.myModule")

    output:
    file("${params.myModuleOutput}") into myModuleOutputChannel
    shell:
    '''
    !{myModuleWorkflow} -params-file !{paramsFile}
    '''
}
```

Containerisation

- Single docker image containing all software dependencies
  - Less configuration burden; longer build process
  - Might migrate to separate images
- All non-trivial tasks in docker image
- Mount host machine's root volume at `/host`
  - Allows all paths to be visible (e.g. shared static data directories) and never any clashes
  - Stage input files with 'rellink' – files use relative symbolic links that work both inside and outside the docker image

nextflow.config

```
docker {
    enabled = true
    fixOwnership = true
    runOptions = "-v \"/:/host/\""
}

process {
    withLabel: bioenv {
        container = 'r-base:3.5.1'
        stageInMode = 'rellink'
    }
}
```

run_config.sh

```
nextflow \
    -C nextflow.config \
    run my_pipeline.nf \
    -params-file my_params.yaml
```

R, Python, Awk, Bash, …

- Nextflow allows you to use the right tool for the right job – use it!
  - Visualise statistical analyses with R
  - Quickly filter data in awk
  - Build TensorFlow models in Python
  - Pretend it's 1995 in Perl
- Works seamlessly with containers
- Remember to properly escape and/or join your input values
  - Don't assume your input will never have spaces or other awkward characters

test_R.nf

```
inChannel = Channel.fromPath("data*.tsv")

process rProcess {
    label 'bioenv'

    input:
    file myFiles from inChannel.toList()

    output:
    file "summary.tsv" into outChannel

    shell:
    '''
    #!/usr/bin/env Rscript

    files_vector <- c("!{myFiles.join('", "')}")

    library("readr")
    :::
}
```

split*, map, group*

- splitText, splitCsv, splitFasta allow for enormous parallelisation – use them!
  - Allow tasks that operate on individual records to use all available hardware
- Retain chunk number for traceability, reproducibility and re-ordering (if required)
- The groupBy and groupTuple operators facilitate recombining outputs.

split_map.nf

```
chunkedChannel = fastaChannel
    .splitFasta(by: 50, file: true)
    .map { chunkFile ->
        regex = "^.*\\.([0-9]+)\\.fasta\$"
        assert chunkFile.name =~ regex
        chunkNum = \
            chunkFile.name.replaceAll(regex, "\$1")
        return [ chunkFile, chunkNum ]
    }

process some_process {
    input:
    set(file(inFile), val(num)) from chunkedChannel
    output:
    set(file("out_${num}"), val(num)) into procChannel
    ...
}

combinedChannel = procChannel
    .groupTuple(by: 1)
    .flatMap { it2 -> [ it2[1].sort { it[0] }.collect {
it[1] } ] }
```

assert, ifEmpty

- Fewer assumptions = more flexible code
  - But also more unwieldy
- When you make an assumption, check that it's valid.
  - Save time and money debugging and running workflows that appear to be working but aren't by using assertions.
- Groovy and Nextflow provide two very useful functions:
  - `assert EXPRESSION`
  - `ifEmpty { CLOSURE }`

assert.nf

```
chunkedChannel = Channel
    .fromPath("*.txt")
    .flatMap { item ->
        numLines = 0;
        item.eachLine { numLines ++ };
        assert numLines > 0 : "input not empty";
        item
    }
    .ifEmpty { error "No input files" }
```

Map-based channels

- Frequently need to keep multiple files/variables together
  - What starts out as a two-item set quickly becomes 8
- Managing the ordering of items, especially for shared/duplicated channels becomes error prone.
- Use map-based channels
  - Name items within channel
  - Add and remove items without major refactoring
  - Throw exception when not available
- Track channel contents with the map function

split_map.nf

```
indexedPatientsChannel = Channel
    .from({ ->
        patientIdx = 0
        params.Patients.collect { patient ->
            patient.patientIdx = patientIdx++
            return [
                'patientIdx': patient.patientIdx, 'patientData': patient
            ]
        }
    }.call())

indexedPatientsChannel.into {
    hlaTypingInputChannel;
    sampleIndexingInputChannel
}

indexedSamplesChannel =
    sampleIndexingInputChannel
    .map { indexedPatient ->
        indexedPatient.patientData.Samples.collect { sample ->
            return [
                'patientIdx': indexedPatient.patientIdx,
                'patientData': indexedPatient.patientData,
                'sampleData': sample
            ]
        }
    }
    .flatMap()
```

Continuous Integration in Nextflow

Unit Testing

- Essential for defining and controlling behaviour.
  - Quickly, frequently catches small unforeseen changes.
- Many platforms available, we use Bitbucket Pipelines.
- Different languages have their own tools – testthat, pytest, JUnit, Test::More, …
  - Can standardise testing and reporting using the Test Anything Protocol
  - Can also be used to lint your codebase
- *Ideally should test specific functions*
- Unit testing Nextflow processes possible:
  - Use the storeDir directive
  - Prepare test/empty files for other processes
  - Run workflow with the -resume flag
  - Test the output

Integration Testing

- Run entire modules
  - Check for behaviour at interfaces between modules
  - Check for characteristics of output (specific details likely to change)
- We have developed framework:
  - Component for preparing test environment (data, repo, job directory)
  - Component for running each test (prepare test directory with data, repo, status monitor)
  - Component for monitoring running tests
- Re-use existing standards, methods (TAP + unittest, pytest)

Regression Testing

- Large tests on multiple full-size datasets
  - Long-running and expensive
- Important part of release validation
  - Monitor and control changes between pipeline versions
- Framework:
  - Defined test datasets. Combinations of ‘real’ and synthetic data for different aspects.
  - Defined tests. Sets of scripts for comparing properties of two pipeline outputs.
  - Execution environment. AWS lambda launches upon test configuration upload.
- Three stage process:
  - Define acceptance criteria. What type and scale of changes are acceptable.
  - Run the test datasets and upload results to commit-hash directory.
  - Upload test configuration and wait for results.

Conclusions

- Nextflow makes writing modern, large-scale pipelines easy and fun.
- Care should still be taken to be defensive.
- There are growing pains associated with building large code bases, but solutions are possible.
- A multi-layered testing framework can detect problems early and save a lot of reviewing and debugging time.
- Validation takes a pipeline from “just another script” to “this thing works”.

Acknowledgements

- Everyone at Achilles, particularly
  - Fionnuala Patterson
  - Michael Epstein
  - FongChun Chan
  - Matthew Davies
  - Khushbu Agrawal
  - Max Salm
  - Gareth Wilson
- TRACERx Consortium
- TRACERx patients
- Nextflow developers and community

l.goodsell@achillestx.com

https://bitbucket.org/achillestx/