Y-chromosomal analysis tools: logic flow document

Abstract

The aim of this programme is to take literature and consumer Y-DNA data from genetic testing companies, and appropriate meta-data from individuals' genealogical histories, and translate them into a geographically and temporally encoded phylogenic tree.

This document describes the coding logic behind the workflow. It is an evolving document designed to communicate and expedite the workflow, rather than a formal description of how the processes work.

Author

Iain McDonald, 26 Jan 2018

Key

Standard flowchart terminology is used for the most part. In addition:
RED items are things that need attention
PURPLE items are data obtained from other sources
YELLOW items are queries to the file system
GREEN items are database entries
BLUE items are storage lists kept in RAM by the programme

Overall programme structure

There are five main challenges to overcome, corresponding to the six work pages listed:
Work Package A: How can we take this diverse dataset and create a manageable and standardised series of calls from it?
Work Package B: How can we parse these calls to create the best haplogroup tree?
Work Package C: Can we chart the progression of Y-STR mutations and perform haplogroup predictions on testers with or without any non-NGS SNP testing?
Work Package D: Can we map the movement of people across Europe using the paper origins of individuals and ancient DNA results?
Work Package E: Can we assign ages to these haplogroups (and by extension the migrations they trace)? How well can we make these line up with known historical events? How can we interface these outputs with those of others?
Work Package F: What can we provide by way of informative and useful outputs?

Work Package A
The primary difficulties here involve managing the large variety of possible input files, the diversity of their respective information, and the differences in their quality standards. The large disc space of these datasets is also an issue, so clearly unimportant variant candidates need to be carefully rejected first. Typical datasets might include:
FASTQ - raw sequences of reads that need aligned to a reference genome (e.g. GRCh38), sources may include literature data or ancient DNA. These may be large fractions of a GB in size.
BAM - aligned data sequences that need called for variants. Sources may include raw BigY data and are typically ~500 MB.
BigY Build 38 VCF - variant call files from the original BAMs. These should have accompanying coverage files (BED) in which regions we are ok to say a variant is passed. The BED regions also define the count for the age analysis. Typical sizes are ~15 / 80 MB zipped/unzipped, and we anticipate thousands of these files.
BigY Build 37 - any legacy data from the old Build 37 data (mapped to GRCh37 / hg19). BAM files can be split back to FASTQ files and remapped to Build 38. VCFs can be processed using "liftOver", which is a simpler co-ordinate conversion. However, this does not allow for better mapping of reads in Build 38, so is not ideal. Size is 1-2 MB zipped.
FGC YElite / WGS - BED files need generated for these using callableLoci in GATK. This is something FGC can do. Variants are stored in gtype files.
YSeq WGS - the Y chromosomal (chrY) data needs extracted from the results, then processed. Status of VCF/BED extraction is currently unclear.
BigY CSV, SNP pack tests, individual YSeq/FTDNA SNP tests, Chromo2 and literature data - these are essentially a list of variants for which a test is positive. Sometimes there are lists of negatives too. Rarely are the information on negative versus null. These data are unsuitable for some analyses (e.g. age analysis) and have limited input into the haplotree, where they would be expected to be added last.

Variants from these inputs will need to be ingested for later comparison, along with associated meta-data like surnames, ancestral origins and STR results. Other STR results, without SNP results, are desirable for later geographical analysis, where they will provide additional datapoints.

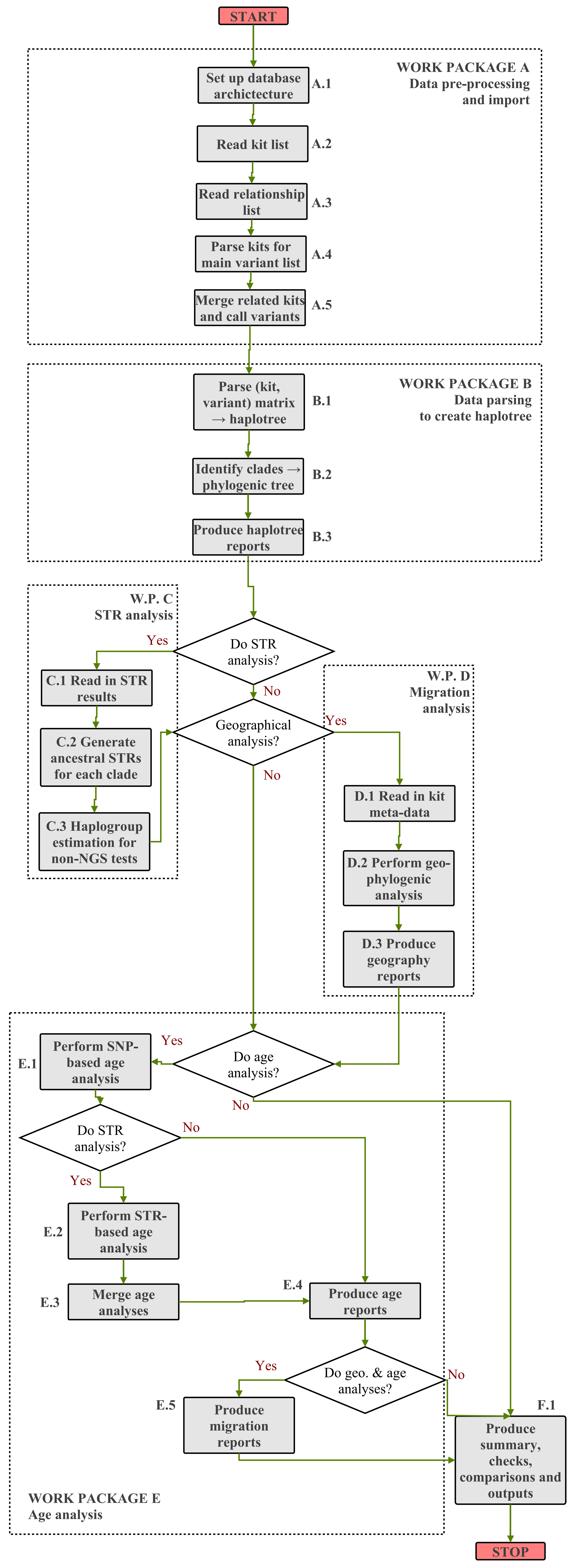
Work Package B
Once ingested and homogenised, this data needs to be parsed to form a haplotree. The difficulties here are minimising the amount of data that needs processed at one time, in order to reduce RAM requirements; and dealing with low quality and missing (uncalled) results.
Parsing will typically require sorting the data into groups of people and clades (haplogroups) of variants to form "blocks". Missing calls will either need to be presumed positive or negative, based foremost on logic, but also on low-quality results in NGS data.
The resulting tree structure is used as input to WPs C, D and E, each of which needs to be capable of running from this input alone, or by including data from the others. Other outputs may compare the output haplotree to those from other sites (e.g. FTDNA's haplotree, ISOGG) to identify where updates need to be made.

Work Package C
This package centres around making use of STR results: both those from the NGS tests and STR-specific tests at the various companies (predominantly FTDNA but also YSeq). Using the NGS test results, we can attempt generation of the ancestral STRs for that clade. This will let us see how STRs have varied across history, and identify the key mutations present in each haplogroup. A further stage of this would be haplogroup estimation for non-NGS-tested individuals.
Not all haplogroups will have a corresponding, stable, identifiable STR mutation. Hence this is only going to be possible for some levels in the tree, and must be an approximate science. Haplogroup estimation can therefore only be done in an approximate sense, but it should provide enough useful data to use in WPs D and E.

Work Package D
This package tries to connect haplogroups to migrations by looking at how the distribution of testers within that haplogroup deviates from the weighted average. This must be done on a clade-by-clade basis to avoid founder effects (e.g. the origin of R-M269 is in Russia, but most R-M269 is now in Europe, due to a founder effect at the R-L11 level).
Difficulties here involve getting a sufficient number of diverse members for each clade (hence why the haplogroup estimation in WP C is important). The data are also heavily biased, since some European regions (e.g. the British Isles) have populations and diaspora that are much more widely tested than other countries (e.g. France, Eastern Europe). Weighting entries to debias them is important, as well as correctly weighting sub-clades and ancient DNA results.

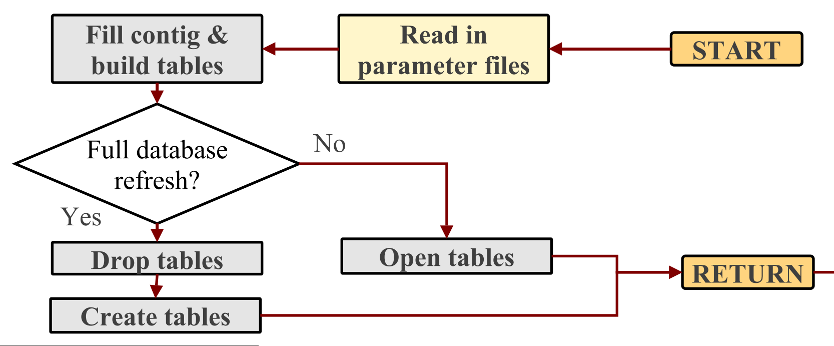
Work Package E
This package performs the age analysis, to put each haplogroup in an historical context. The basic principles behind this are effectively worked out, but take a lot of CPU power. STR-based age analysis can be done fairly easily, but the results will need carefully calibrated and cross-checked for consistency. Treatment of uncertainties is also very important. Most of the problems here are likely to be mathematical in nature.

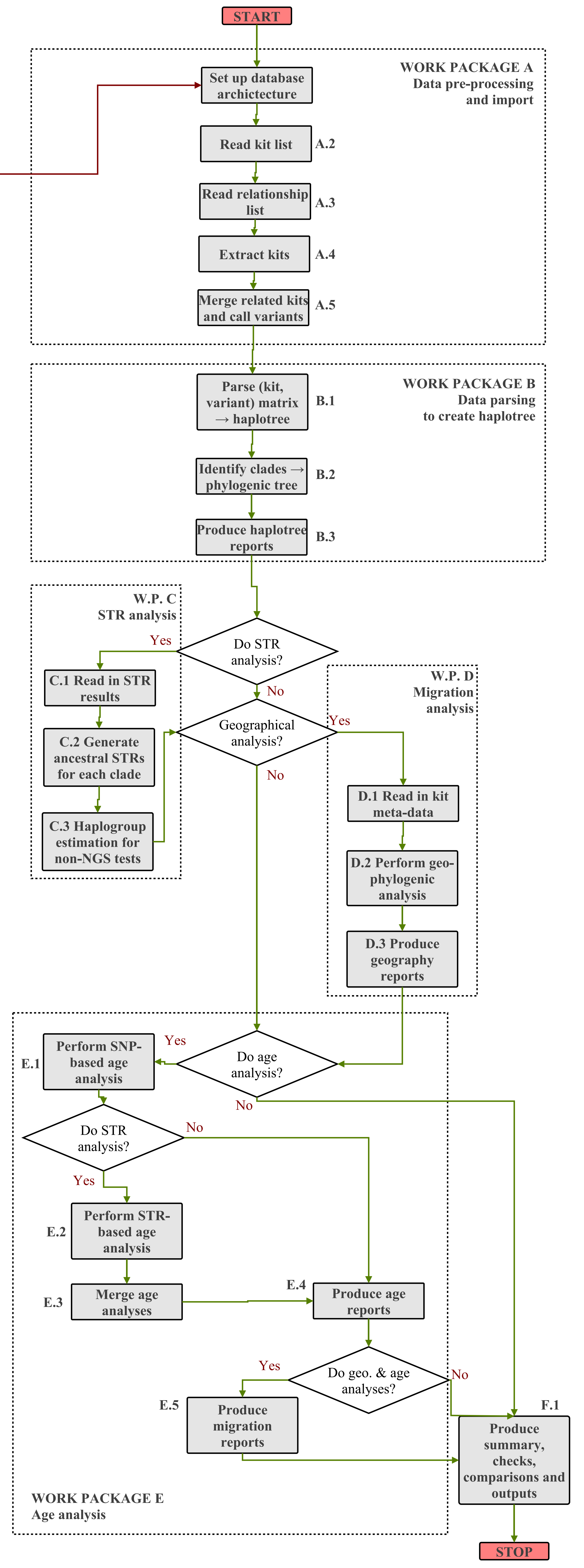
Work Package F
A range of outputs from this data can be considered, from a list of ages and co-ordinates, to bespoke queries on individual tests (haplogroup estimation, STR mutation timeline, migration pathway), to animations of migrations across Europe.


START
WORK PACKAGE A
Data pre-processing and import
Set up database architecture
A.1
Read kit list
A.2
Read relationship list
A.3
Parse kits for main variant list
A.4
Merge related kits and call variants
A.5
WORK PACKAGE B
Data parsing to create haplotree
Parse (kit, variant) matrix → haplotree
B.1
Identify clades → phylogenic tree
B.2
Produce haplotree reports
B.3
W.P. C
STR analysis
Do STR analysis?
Yes
No
C.1 Read in STR results
C.2 Generate ancestral STRs for each clade
C.3 Haplogroup estimation for non-NGS tests
Geographical analysis?
Yes
No
W.P. D
Migration analysis
D.1 Read in kit meta-data
D.2 Perform geo-phylogenic analysis
D.3 Produce geography reports
Do age analysis?
Yes
No
E.1
Perform SNP-based age analysis
Do STR analysis?
No
Yes
E.2
Perform STR-based age analysis
E.3
Merge age analyses
E.4
Produce age reports
Do geo. & age analyses?
Yes
No
E.5
Produce migration reports
F.1
Produce summary, checks, comparisons and outputs
WORK PACKAGE E
Age analysis
STOP

Item A.1: Database architecture

This step sets up the initial database structures for the programme (or imports them), along with any parametric setup files and data pointers.
Uses global parameters: `ResetDB` `Boolean:` Drop all tables and remake database
External inputs: User parameter file




START
WORK PACKAGE A
Data pre-processing and import
Set up database architecture
Read kit list
A.2
Read relationship list
A.3
Extract kits
A.4
Merge related kits and call variants
A.5
WORK PACKAGE B
Data parsing to create haplotree
Parse (kit, variant) matrix → haplotree
B.1
Identify clades → phylogenic tree
B.2
Produce haplotree reports
B.3
W.P. C
STR analysis
Do STR analysis?
Yes
No
C.1 Read in STR results
C.2 Generate ancestral STRs for each clade
C.3 Haplogroup estimation for non-NGS tests
Geographical analysis?
Yes
No
W.P. D
Migration analysis
D.1 Read in kit meta-data
D.2 Perform geo-phylogenic analysis
D.3 Produce geography reports
Do age analysis?
Yes
No
E.1
Perform SNP-based age analysis
Do STR analysis?
Yes
No
E.2
Perform STR-based age analysis
E.3
Merge age analyses
E.4
Produce age reports
Do geo. & age analyses?
Yes
No
E.5
Produce migration reports
F.1
Produce summary, checks, comparisons and outputs
WORK PACKAGE E
Age analysis
STOP

Database tables

The following tables describe the database structures. Grey fields are currently unused. Red fields are not in the current (24 Jan 2018) schema.

PERSON – primary list of people

Name	*Type*	*Description*	*Filled in*	*Used in*
ID	INTEGER PRIMARY KEY	-	A.4	-
firstName	TEXT DEFAULT NULL	Name of tester	A.4	-
middleName	TEXT DEFAULT NULL	Name of tester	A.4	-
surname	TEXT DEFAULT NULL	Name of tester	A.4	-
maidenName	TEXT DEFAULT NULL	Name of tester	A.4	-
yHaplogroupId	INTEGER DEFAULT NULL	Most-recent Y-DNA hg	B	-
mtHaplogroupId	INTEGER DEFAULT NULL	Most-recent mt-DNA hg	-	-

DATABASE – primary list of kits

Name	*Type*	*Description*	*Filled in*	*Used in*
ID	INTEGER PRIMARY KEY	-	A.2	-
seq	INTEGER DEFAULT 0	Order files arbitrarily	B	-
kitName	TEXT	A name for the dataset	A.2,A.4	-
DNAID	INTEGER references person(ID)	Whose DNA is this?	A.2,A.4	A.4
contact	TEXT	Holder's contact info	A.2	-
kitId	TEXT [INDEX fileidx]	Lab ID	A.2	-
surnameID	INTEGER references surname(ID)	MDKA surname	A.2,A.4	-
countryID	INTEGER references country(ID)	MDKA country	A.2,A.4	-
birthYr	INTEGER DEFAULT NULL	MDKA birth year	A.2,A.4	-
labID	INTEGER references lab(id)	Testing lab	A.2	-
testTypeID	INTEGER REFERENCES testtype(ID)	Type of DNA test	A.2	-
buildID	INTEGER references build(ID)	Reference build for the test	A.2	-
importDt	TEXT	Date originally imported	A.2	-
fileNm	TEXT	Normalized file/path name	A.2	-
origFileNm	TEXT	Original file name from user	A.2	-
otherInfo	TEXT	Freeform text entered by user	A.2,A.4	-
normalOrigID	INTEGER references origin(ID)	Normalized ancestral origin	A.2,A.4	-
lat	TEXT	Ancestral location latitude	A.2,A.4	-
lng	TEXT	Ancestral location longitude	A.2,A.4	-
policyVer	TEXT DEFAULT NULL	Data policy agreed to	A.2	-
accessToken	TEXT DEFAULT NULL	-	-	-
updated	TEXT	Date the record was updated	A.2	-
approxHg	TEXT	Haplogroup assigned by DW	A.2	-

SURNAME – list of surnames

Name	*Type*	*Description*	*Filled in*	*Used in*
ID	INTEGER PRIMARY KEY	-	A.2	-
surname	TEXT	Name	A.2	-

TESTTYPE – list of tests

Name	*Type*	*Description*	*Filled in*	*Used in*
ID	INTEGER PRIMARY KEY	-	A.2	-
testNm	TEXT	The name of the test	A.2	-
description	TEXT	Further description	A.2	-
isNGS	BOOLEAN	Is it a sequencing test?	A.2	-
tagNm	TEXT	Field in hg-R schema	-	-
priority	INTEGER	Field in hg-R schema	-	-

CONTIG – defines regions of interest

Name	*Type*	*Description*	*Filled in*	*Used in*
ID	INTEGER PRIMARY KEY	-	A.1	-
buildID	INTEGER references build(ID)	Reference build, e.g. hg38	A.1	-
description	TEXT	e.g. chrY	A.1	-
length	INTEGER	e.g. 57227415	A.1	-

COUNTRY – list of countries

Name	*Type*	*Description*	*Filled in*	*Used in*
ID	INTEGER PRIMARY KEY	-	A.2	-
country	TEXT	Name	A.2	-

ORIGIN – list of origins

Name	*Type*	*Description*	*Filled in*	*Used in*
ID	INTEGER PRIMARY KEY	-	A.2	-
origin	TEXT	Name	A.2	-

LAB – list of labs

Name	*Type*	*Description*	*Filled in*	*Used in*
ID	INTEGER PRIMARY KEY	-	A.2	-
labNm	TEXT	Name	A.2	-

BEDRANGES – minimum and maximum ranges in BED file entries

Name	*Type*	*Description*	*Filled in*	*Used in*
ID	INTEGER PRIMARY KEY	-	A.5	-
minaddr	INTEGER [INDEX rangeidx]	Position of minimum	A.5	-
maxaddr	INTEGER [INDEX rangeidx2]	Position of maximum	A.5	-

BED – entries present in each BED file

Name	*Type*	*Description*	*Filled in*	*Used in*
pID	INTEGER REFERENCES dataset(ID)	Person	A.5	-
bID	INTEGER REFERENCES bedranges(ID)	BED entry	A.5	-

TREE – description of the haplogroup tree

Name	*Type*	*Description*	*Filled in*	*Used in*
ID	INTEGER PRIMARY KEY	-	B	-

Input is needed on this design, perhaps conforming to an established standard

ALLVARIANTS – positions and counts for every entry present in VCF files

Name	*Type*	*Description*	*Filled in*	*Used in*
pID	INTEGER REFERENCES dataset(ID)	Person	A.4	-
buildID	INTEGER REFERENES build(ID)	Build	A.4	-
pos	INTEGER	Position	A.4	-
refalleleID	INTEGER REFERENCES alleles(ID)	Allele called	A.4	-
deralleleID	INTEGER REFERENCES rules(ID)	Reason called	A.4	-
npos	INTEGER	# positive calls	A.4	-
nneg	INTEGER	# negative calls	A.4	-
nmix	INTEGER	# mixed calls	A.4	-

VCFCALLS – entries present in each VCF file

Name	*Type*	*Description*	*Filled in*	*Used in*
pID	INTEGER REFERENCES dataset(ID)	Person	A.5	-
vID	INTEGER REFERENCES variants(ID) … [INDEX vcfidx]	VCF entry	A.5	-
alleleID	INTEGER REFERENCES alleles(ID)	Allele called	A.5	-
ruleID	INTEGER REFERENCES rules(ID)	Reason called	A.5	-
qual	FLOAT	Quality	A.5	-

VCFSTATS – reported statistics on VCF files

Name	*Type*	*Description*	*Filled in*	*Used in*
pID	INTEGER REFERENCES dataset(ID)	Person	B	-
ny	INTEGER	-	B	-
nv	INTEGER	-	B	-
ns	INTEGER	-	B	-
ni	INTEGER	-	B	-
nr	INTEGER	-	B	-

BEDSTATS – reported statistics on BED files

Name	*Type*	*Description*	*Filled in*	*Used in*
pID	INTEGER REFERENCES dataset(ID)	Person	A.5	-
coverage1	INTEGER	Total coverage	A.5	-
coverage2	INTEGER	Cov. for age analysis	A.5	-
nranges	INTEGER	# entries	A.5	-

META – meta-data for the entire data set or run

Name	*Type*	*Description*	*Filled in*	*Used in*
desc	TEXT	-	-	-
val	TEXT	-	-	-

STRS – table for STR definitions

Name	*Type*	*Description*	*Filled in*	*Used in*
ID	INTEGER PRIMARY KEY	-	C	-
strname	TEXT	Name	C	-
ordering	INTEGER	Put STRs in order	C	-

STRCALLS – tester's calls for STRs

Name	*Type*	*Description*	*Filled in*	*Used in*
pID	INTEGER REFERENCES people(ID)	Person	C	-
sID	INTEGER REFERENCES strs(ID)	STR	C	-
val	INTEGER	Allele	C	-

VARIANTS – list of known variants [INDEX varidx on buildID,pos]

Name	*Type*	*Description*	*Filled in*	*Used in*
ID	INTEGER PRIMARY KEY	-	A.5	-
buildID	INTEGER REFERENCES build(ID)	Build	A.5	-
pos	INTEGER	Position	A.5	-
anc	INTEGER REFERENCES alleles(ID)	Ancestral allele	A.5	-
der	INTEGER REFERENCES alleles(ID)	Derived allele	A.5	-

ALLELES – list of alleles in sequencing calls

Name	*Type*	*Description*	*Filled in*	*Used in*
ID	INTEGER PRIMARY KEY	-	A.5	-
allele	TEXT	e.g. A,C,G,T,AT,…	A.5	-

SNPNAMES – names and aliases associated with variants

Name	*Type*	*Description*	*Filled in*	*Used in*
vID	INTEGER REFERENCES variants(ID)	-	A.5	-
snpname	TEXT	e.g. U106	A.5	-

BUILD – list of reference genome assemblies used

Name	*Type*	*Description*	*Filled in*	*Used in*
ID	INTEGER PRIMARY KEY	-	A.1	-
buildNm	TEXT	-	A.1	-

AGEBED – restricted list of ranges that are used to calculate the ages of clades

Name	*Type*	*Description*	*Filled in*	*Used in*
ID	INTEGER PRIMARY KEY	-	A.5	-
bID	INTEGER REFERENCES bedranges(ID)	-	A.5	-

Item A.2: Read in list of kits

This step identifies whether any new kits are to be read in.

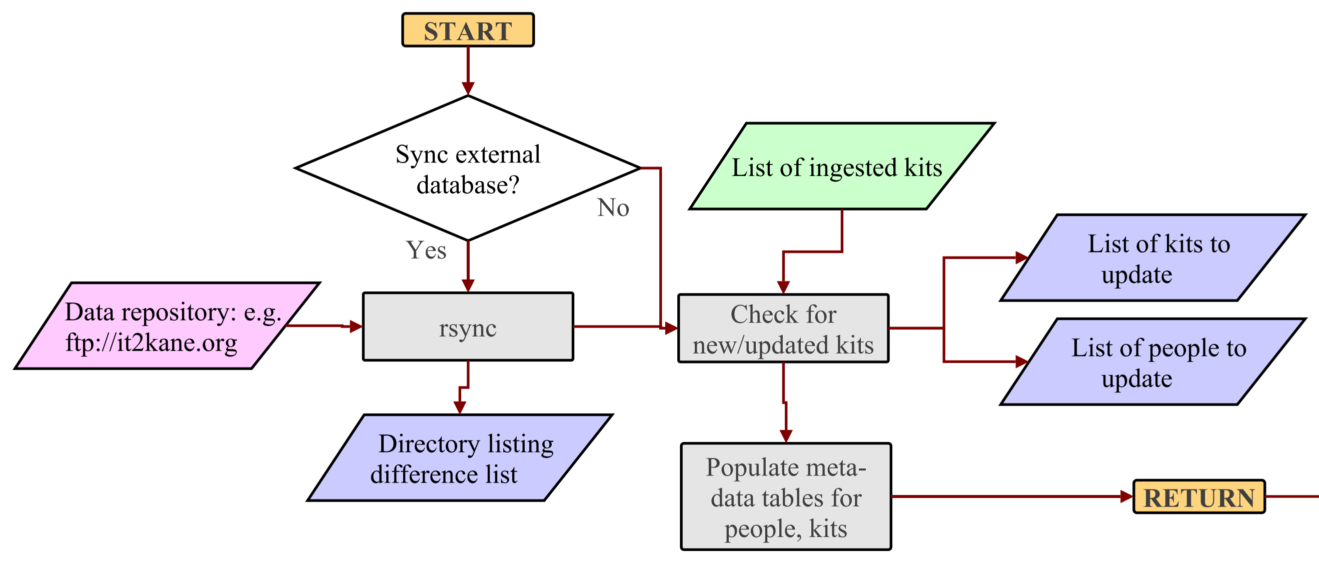
Uses global parameters:

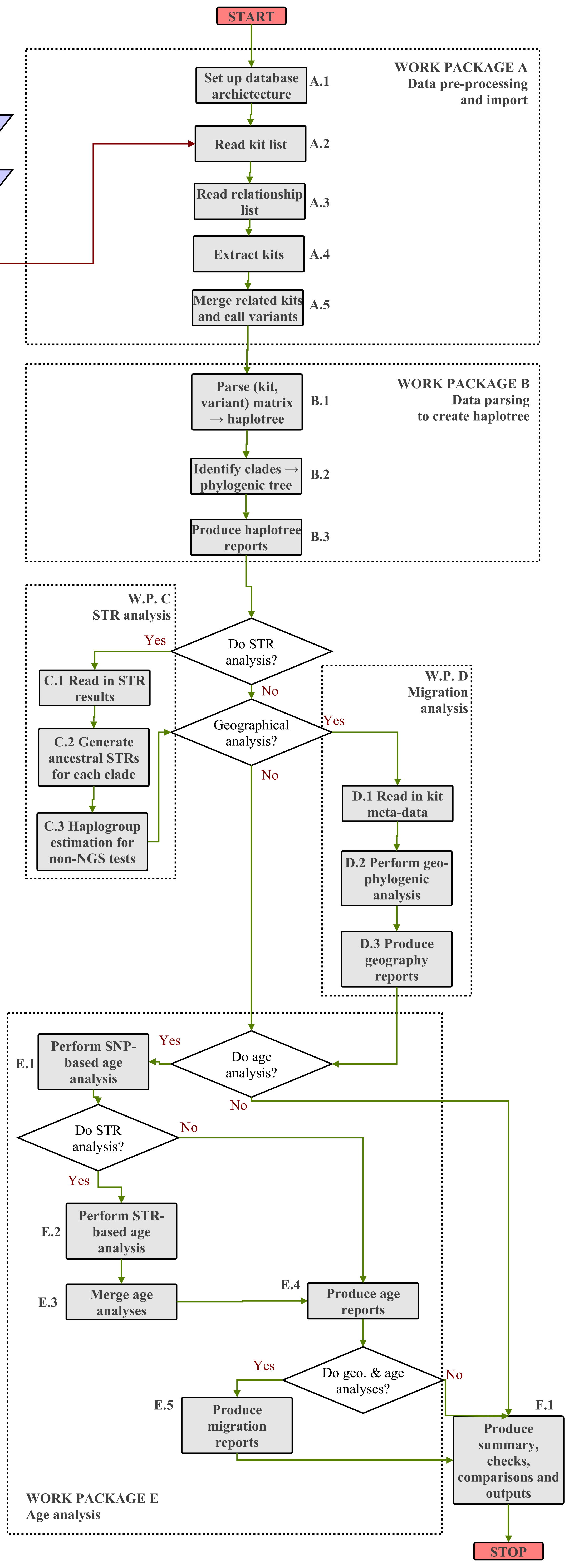
`SyncToDB`	Boolean: synchronise to external database?

External inputs:
Data repository

Internal inputs:
Internal database **`kits`**
Internal database **`people`**

Produces:
List of **`kits`** to update
List of **`people`** to update


START
Sync external database?
No
Yes
Data repository: e.g. ftp://it2kane.org
rsync
Directory listing difference list
List of ingested kits
Check for new/updated kits
List of kits to update
List of people to update
Populate meta-data tables for people, kits
RETURN


START
WORK PACKAGE A
Data pre-processing and import
Set up database archictecture
A.1
Read kit list
A.2
Read relationship list
A.3
Extract kits
A.4
Merge related kits and call variants
A.5
WORK PACKAGE B
Data parsing to create haplotree
Parse (kit, variant) matrix → haplotree
B.1
Identify clades → phylogenic tree
B.2
Produce haplotree reports
B.3
W.P. C
STR analysis
Do STR analysis?
Yes
No
C.1 Read in STR results
C.2 Generate ancestral STRs for each clade
C.3 Haplogroup estimation for non-NGS tests
Geographical analysis?
Yes
No
W.P. D
Migration analysis
D.1 Read in kit meta-data
D.2 Perform geo-phylogenic analysis
D.3 Produce geography reports
Do age analysis?
Yes
No
E.1
Perform SNP-based age analysis
Do STR analysis?
No
Yes
E.2
Perform STR-based age analysis
E.3
Merge age analyses
E.4
Produce age reports
Do geo. & age analyses?
Yes
No
E.5
Produce migration reports
F.1
Produce summary, checks, comparisons and outputs
WORK PACKAGE E
Age analysis
STOP

Item A.3: Read in list of relationships

This list identifies and describes the relationships between different kits. Applications are twofold.

Firstly, kits belonging to the same person need to be merged for best effect (A.4). Kits belonging to closely related individuals (e.g. father/son or cousin pairs) may be best merged as well.

Secondly, known relationships become important when performing the age analysis, in defining limits to relationships. The same list can be used to define the ages of ancient DNA samples.

The input list may be spread over one or more files, and contain three data types:

- MDKA: provides a date (with or without uncertainty) for the kit holder's most-distant known ancestor's birth. Relationships cannot be younger than the latter of two tests' MDKAs.
- DoB: provides a date of birth for a tester, or a probability distribution function (PDF) describing this. This will mainly be useful for carbon-14 dates from ancient DNA results, which may come as arbitrarily complex PDFs.
- Link: a known common ancestor links two tests. This may be two tests from the same person. It may have some uncertainty, and it may be an arbitrarily complex function (e.g. for testers known/suspected to have a common surname origin or emigrant history). It can also be used to set arbitrary boundaries like "clades must be related after the Norman Conquest" or "must be related within the Corded Ware Culture period", etc.

Data formats may be:

`Type=MDKA, Kit, Date, Unc, LowLimit, UpLimit, Comment`

`Kit`	Must be unique identifier (e.g. sequence number or include company)
`Date`	As common epoch years (CE) [default=`zeroage`].
`Unc`	Gaussian uncertainty parameter [default=`zeroage`**`unc`**].
`Low/UpLimit`	Youngest/oldest possible date [default=`currentYear/-inf`].

`Type=DoB, Kit, Date, Unc, LowLimit, UpLimit, PDFfile, Comment`

`PDFfile`	[Default=`null`] Contains arbitrary PDF formatted as:
	`Year     Probability`

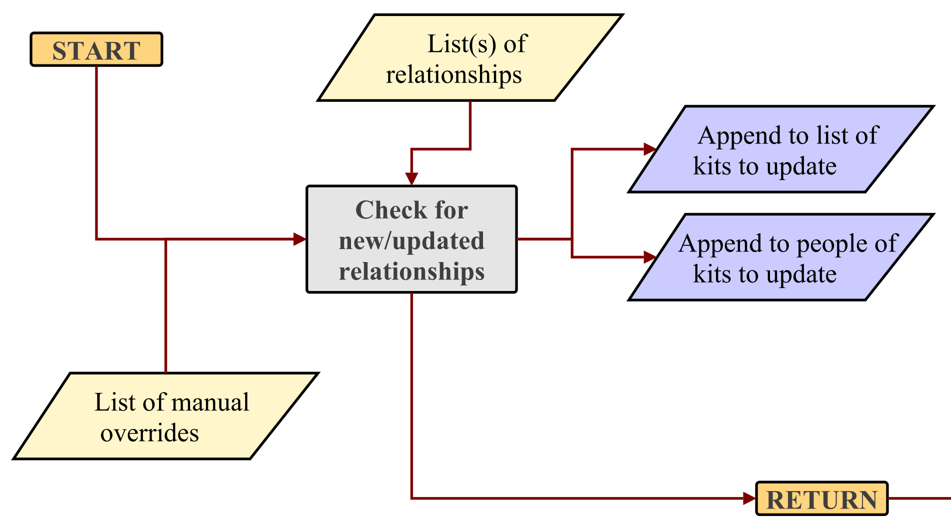
`Type=Link, Kit1, Kit2, Date, Unc, LowLimit, UpLimit, PDFfile, Merge?, Comment`

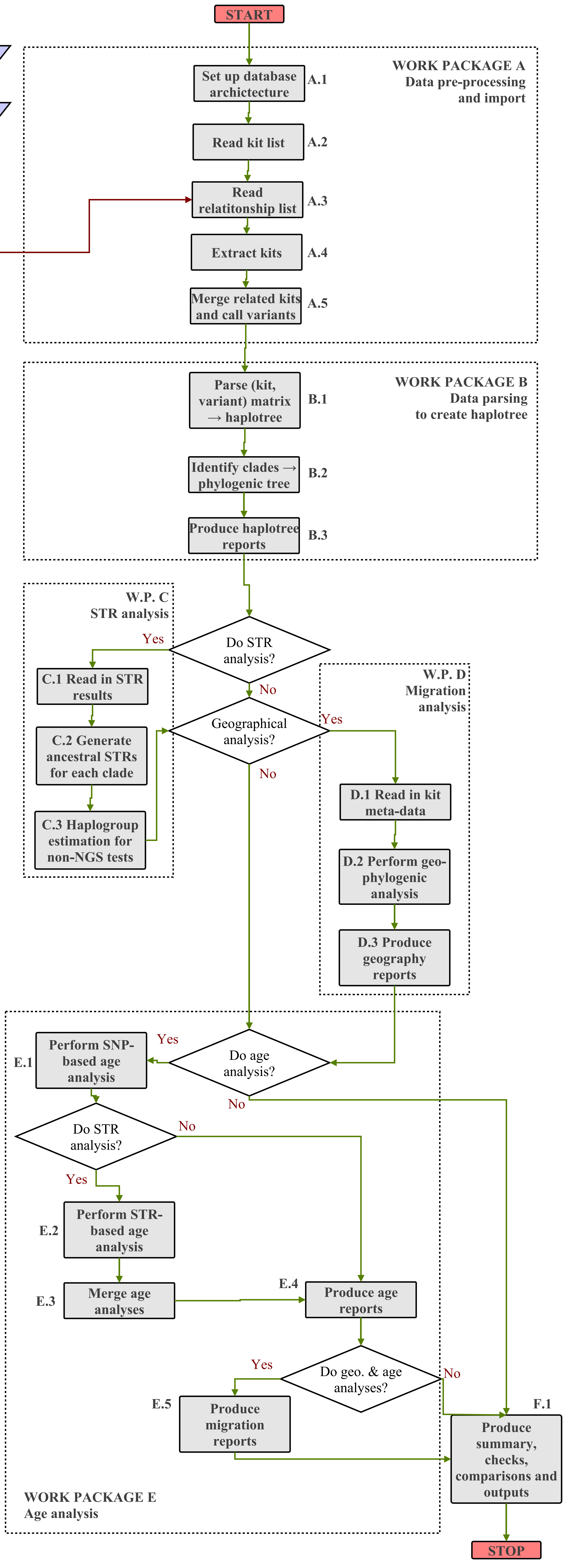
`Kit1,2`	The two kits under consideration
`Merge`	Boolean TRUE/FALSE to merge kits into the same `person` [default=`false`]

Manual over-rides to table entries can be made at this point too. This allows us to set things such as people's ancestral locations and MDKA information with more accurate information that they have provided but is not available from the primary data sources. Data formats may be:
`Table, KitID, PersonID, Field, Data, Comment`
e.g.:
`Kits, 123456, , mdkaDoB, 1700, IM: updated from E-mail`

Uses global parameters:

`RelationFile`	File(s) containing list of relationships
`zeroage`	Default DoB of testers (c. 1950).
`zeroageunc`	Associated uncertainty (c. 16 years).


START
List(s) of relationships
Check for new/updated relationships
Append to list of kits to update
Append to people of kits to update
List of manual overrides
RETURN


START
WORK PACKAGE A
Data pre-processing and import
Set up database architecture
A.1
Read kit list
A.2
Read relatitonship list
A.3
Extract kits
A.4
Merge related kits and call variants
A.5
WORK PACKAGE B
Data parsing to create haplotree
Parse (kit, variant) matrix → haplotree
B.1
Identify clades → phylogenic tree
B.2
Produce haplotree reports
B.3
W.P. C
STR analysis
Do STR analysis?
Yes
No
C.1 Read in STR results
C.2 Generate ancestral STRs for each clade
C.3 Haplogroup estimation for non-NGS tests
Geographical analysis?
Yes
No
W.P. D
Migration analysis
D.1 Read in kit meta-data
D.2 Perform geo-phylogenic analysis
D.3 Produce geography reports
Do age analysis?
Yes
No
E.1
Perform SNP-based age analysis
Do STR analysis?
Yes
No
E.2
Perform STR-based age analysis
E.3
Merge age analyses
E.4
Produce age reports
Do geo. & age analyses?
Yes
No
E.5
Produce migration reports
F.1
Produce summary, checks, comparisons and outputs
WORK PACKAGE E
Age analysis
STOP

Item A.4: Extract kits

This will be one of the harder parts of this work package to get right, due to the volume of data involved and the diversity of data we have to deal with. This diversity is likely to require specific analyses for each kind of data.

As well as BigY, FGC YElite and WGS, and YSeq WGS tests, we may have to deal with arbitrary lists of calls from CSV calls from BigY, National Geographic, Chromo2, YSeq, literature etc. These may be presented as hg36, hg37 or hg38 positions, or as named SNPs.

BAM and FASTQ files from literature sources may include modern or ancient DNA. Initial quality filters for ancient DNA will likely need to be lowered.

Extraneous data (e.g. negative calls, failed calls, quality data) cannot be completely thrown away at this point, because we don't know which variants we're interested in until we've looked at all the ZIP files. However, the unzipped VCF files *may* ultimately become too large to reasonably store on disc. We can either: (1) keep the unzipped files if they're not too large; or (2) remove the unzipped files each time, then re-unzip them again later. Let the user decide via **`keepunzip`**. The logic can be simplified if we don't actually edit the files, but can read them from their ZIPped state.

We are only interested in variants where there are a mixture of positive and negative calls. Every other variant is either shared among everyone (above the tree) or never positive (outside the tree). The first VCF pass selects any variant which has the following: $7=PASS and GT=0/0 (reference call), 1/1 (derived call) or 0/1 (mixed call). These are parsed into an array of:

positions, allele(ref), allele(der), count(ref), count(der), count(mix).

At the end of this process, we can select those variants where (count(positive)>0 || count(mix)>0) && count(negative)>0 as those we want to investigate: these are variants found in someone but not everyone, so are those needed to make the tree. These make the master **`variants`** list, and we assume for now that the reference allele is the ancestral allele (positives in the reference sequence are dealt with in the next item).

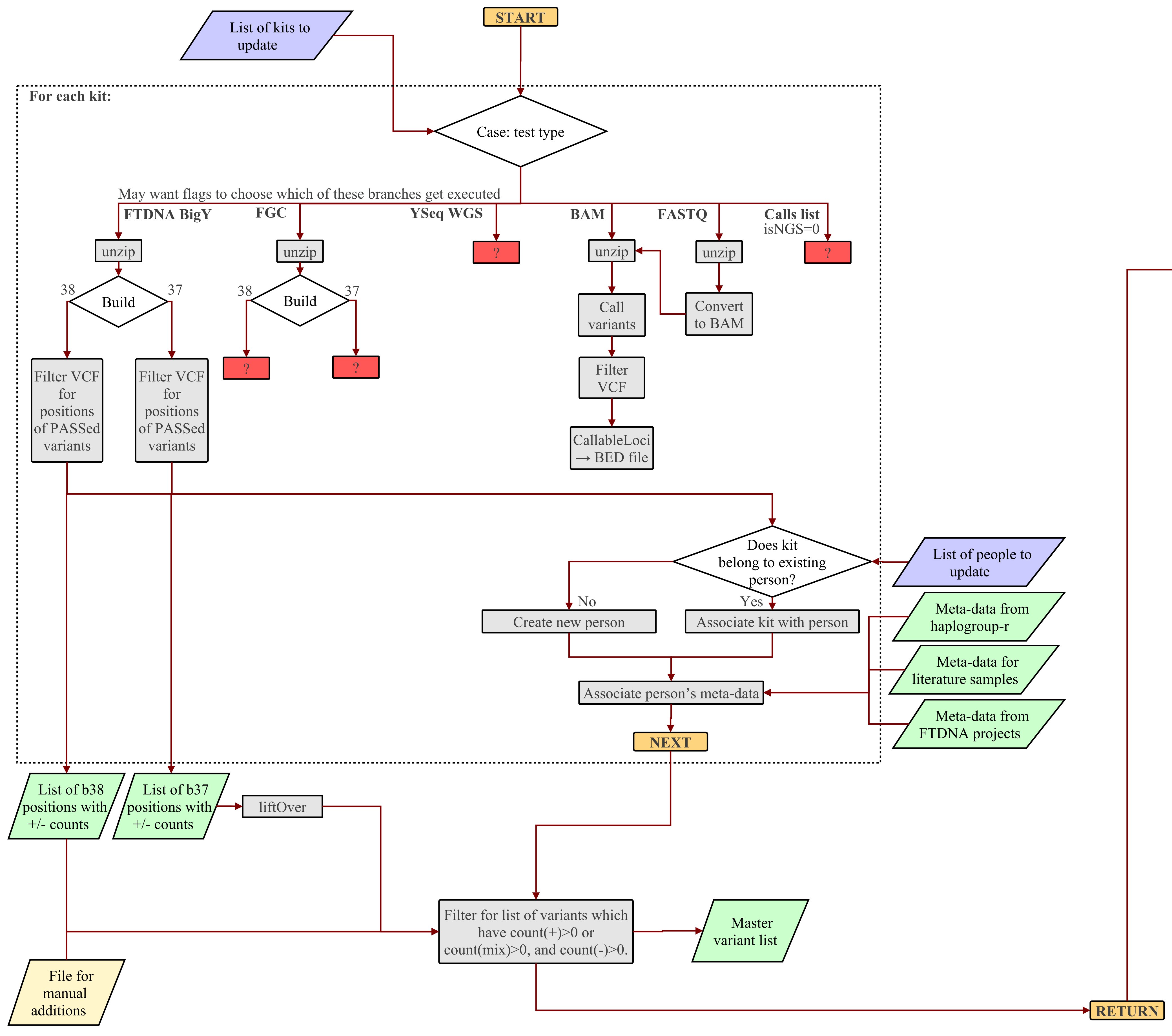
Uses global parameters:

`useBigY`	Boolean: extract new BigY results?
`useFGC`	Boolean: extract new FGC results?
	etc.
`usehg37`	Boolean: extract build 37 results?
`keepunzip`	Boolean: keep or delete unzipped files?

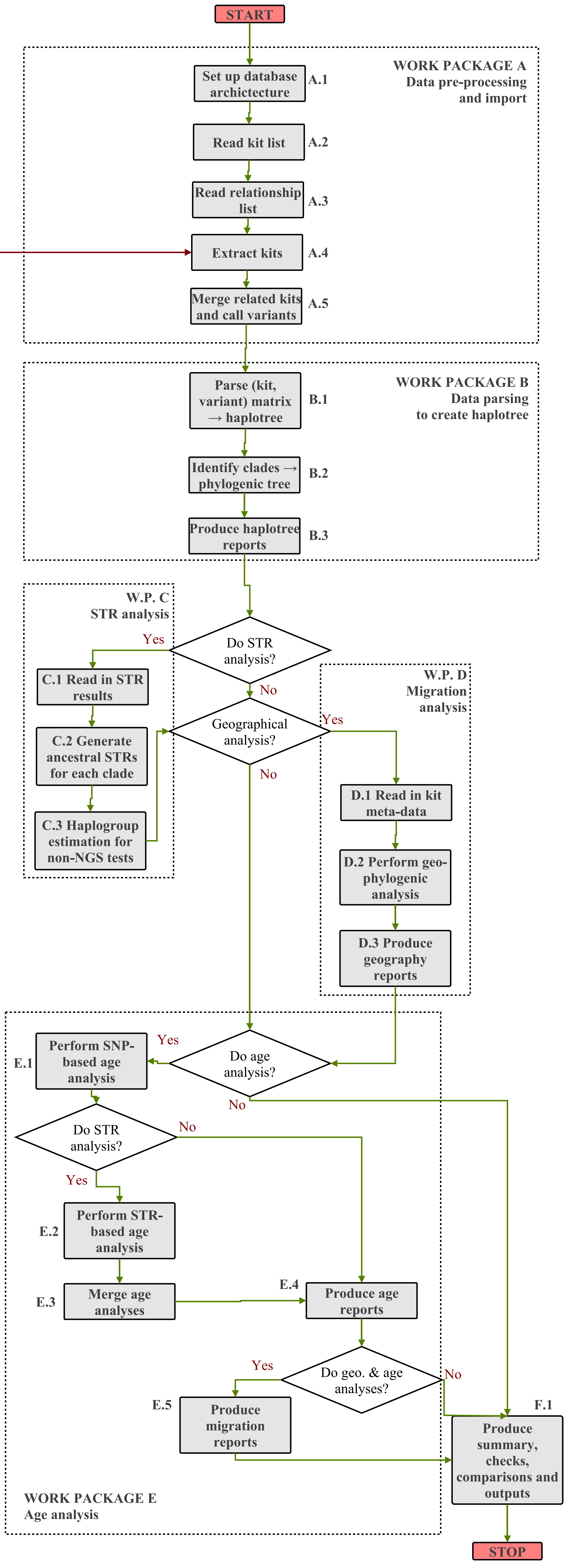
External inputs:
Data repository meta-data

Internal inputs:
Lists of **`kits`** and **`people`** to update

Produces:
Master list of **`variants`**



List of kits to update
START
For each kit:
Case: test type
May want flags to choose which of these branches get executed
FTDNA BigY
FGC
YSeq WGS
BAM
FASTQ
Calls list isNGS=0
unzip
unzip
?
unzip
unzip
?
38
Build
37
38
Build
37
?
?
Call variants
Convert to BAM
Filter VCF
CallableLoci → BED file
Filter VCF for positions of PASSed variants
Filter VCF for positions of PASSed variants
Does kit belong to existing person?
List of people to update
No
Yes
Create new person
Associate kit with person
Meta-data from haplogroup-r
Meta-data for literature samples
Meta-data from FTDNA projects
Associate person's meta-data
NEXT
List of b38 positions with +/- counts
List of b37 positions with +/- counts
liftOver
Filter for list of variants which have count(+)>0 or count(mix)>0, and count(-)>0.
Master variant list
File for manual additions
RETURN



START
WORK PACKAGE A
Data pre-processing and import
Set up database architecture
A.1
Read kit list
A.2
Read relationship list
A.3
Extract kits
A.4
Merge related kits and call variants
A.5
WORK PACKAGE B
Data parsing to create haplotree
Parse (kit, variant) matrix → haplotree
B.1
Identify clades → phylogenic tree
B.2
Produce haplotree reports
B.3
W.P. C
STR analysis
Do STR analysis?
Yes
No
C.1 Read in STR results
C.2 Generate ancestral STRs for each clade
C.3 Haplogroup estimation for non-NGS tests
Geographical analysis?
Yes
No
W.P. D
Migration analysis
D.1 Read in kit meta-data
D.2 Perform geo-phylogenic analysis
D.3 Produce geography reports
Do age analysis?
Yes
No
E.1
Perform SNP-based age analysis
Do STR analysis?
No
Yes
E.2
Perform STR-based age analysis
E.3
Merge age analyses
E.4
Produce age reports
Do geo. & age analyses?
Yes
No
E.5
Produce migration reports
F.1
Produce summary, checks, comparisons and outputs
WORK PACKAGE E
Age analysis
STOP

Item A.5: Merge related kits / call variants

Several kits may be associated with one person. This may include any form of combinations, e.g.: BigY+YElite, BigY+YSeq/WGS, Y-STRs + literature PGP test, Geno 2.0 + BigY, Y-STRs + YSeq SNPs, etc.

One method of dealing with this is to call a list of (people,variants), rather than (kits, variants) to form the haplotree with. Care must be taken to ensure that the most secure determination of a call is taken: e.g., how does a rejected BigY call with BQ=40, MQ=40 and 10/14 reads positive compare with a FGC ** call? This may require careful thought, and the option of guiding the tree structure in part B with a series of manually injected calls may be necessary.

Even if we take a pure (people,variants) list of calls, the merger of two tests must result in a new BED file. This can be a straight merger of data. E.g.:

```
chrY 120000   125000
chrY 127000   128000
```

and

```
chrY 124000   126000
chrY 129000   130000
```

would merge to:

```
chrY 120000   126000
chrY 127000   128000
chrY 129000   130000
```

Where individual SNP calls have been made by packs or individual tests, these entries will need added into the BED file, e.g.:

```
chrY 12060400 12060401
```

for a Z301 call.

At this point, we also try to fix positives in the reference sequence. These occur because the reference sequence is primarily made from the Y-DNA of a R-U152 male, hence is positive for all SNPs from Y-DNA Adam down through the R-U152 sequence and below. YBrowse's SNP names list maintains the ancestral version, so we can use this to correct these reference-sequence positives when their ancestral values don't match our reference sequence values.

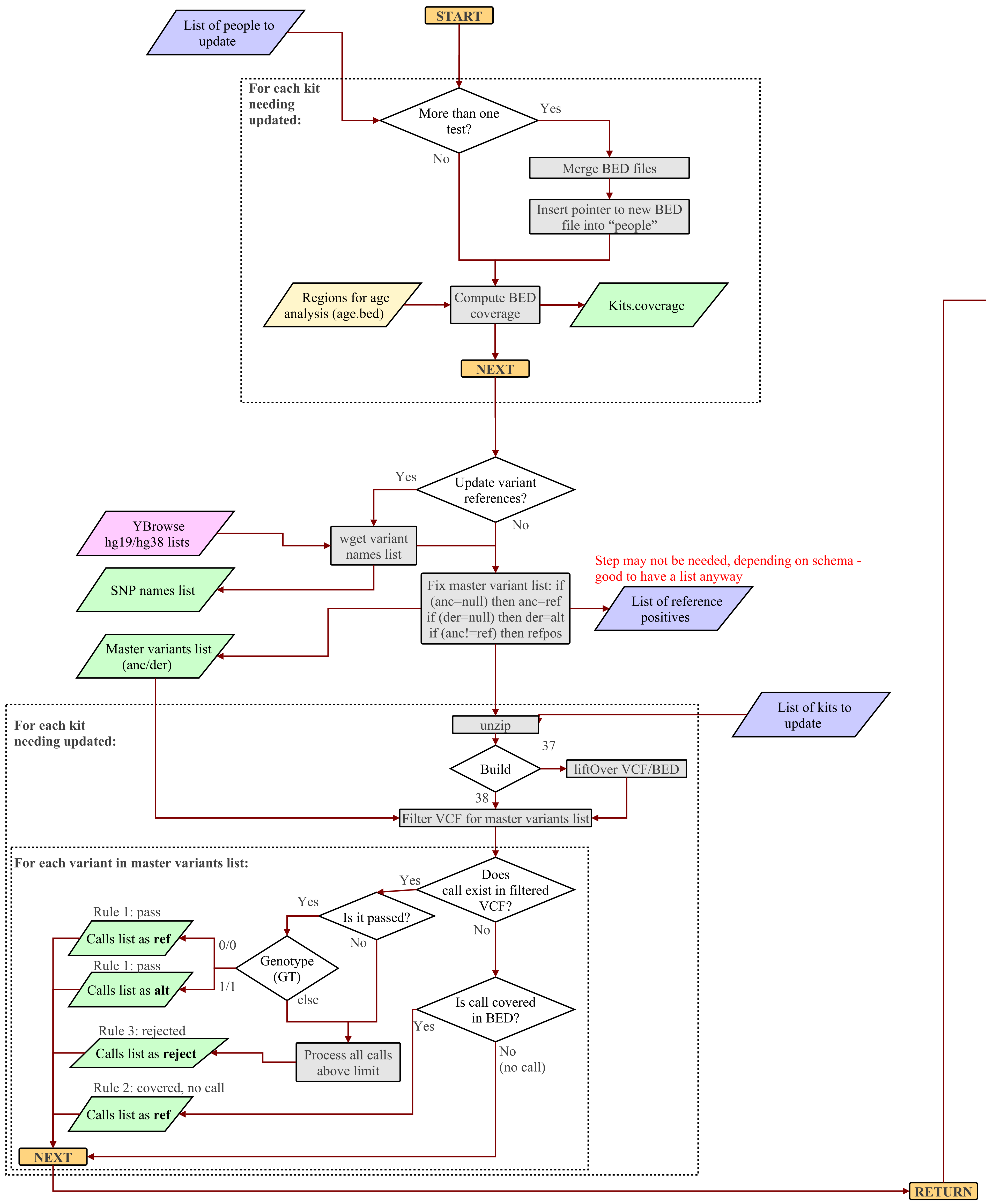
Once this is done, we can make our calls. For calls which pass the quality criteria (field[6] = "PASS" [zero-indexed]), the genotype (field[10,1]) will represent 0/0 for the reference allele, 1/1 for the derived allele, and may be something else (0/1, 1/2, etc.) for a mixture of calls. Note that the derived allele may be comma-separated if there is more than one possible alternative for that call in that test. Hence, all calls need processed.

The quality filter needs recorded: this is -10*log10(Probability call is wrong). So QUAL=10 implies P=0.1, or a 10% chance of a bad call. In a test of 10 million base pairs, QUAL < 70 may be spurious. Variants may also be rejected because the depth (DP) is too high: this could occur if multiple regions of DNA are mapped to the same co-ordinates because they are too similar.

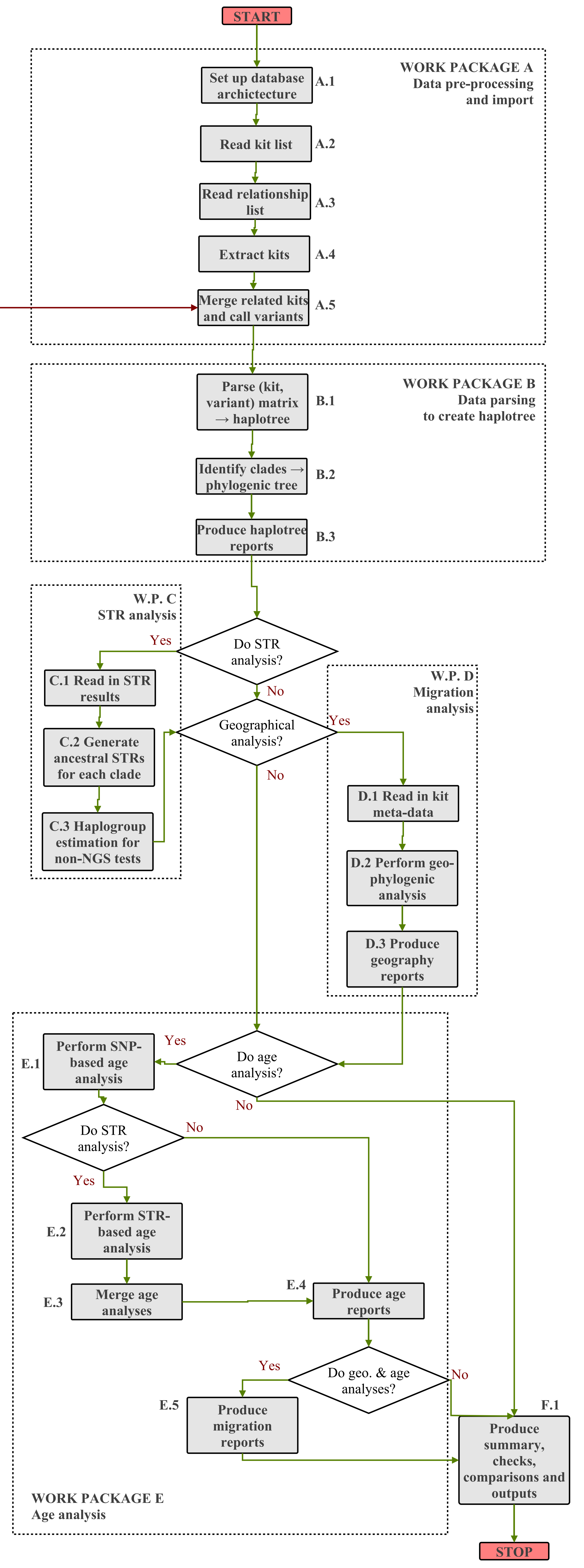
External inputs:
YBrowse's list of variants

Internal inputs:
Master list of **`variants`**
List of **`people`** to update

Produces:
Merged coverage files for people with more than one test
Coverage statistics for each **`kit`**
Reference positive corrections for the **`variants`** list
A list of **`snpnames`**
Set of **`calls`** for each person/variant



START
List of people to update
For each kit needing updated:
More than one test?
Yes
No
Merge BED files
Insert pointer to new BED file into "people"
Regions for age analysis (age.bed)
Compute BED coverage
Kits.coverage
NEXT
Update variant references?
Yes
No
YBrowse hg19/hg38 lists
wget variant names list
SNP names list
Fix master variant list: if (anc=null) then anc=ref if (der=null) then der=alt if (anc!=ref) then refpos
Step may not be needed, depending on schema - good to have a list anyway
List of reference positives
Master variants list (anc/der)
For each kit needing updated:
List of kits to update
unzip
Build
37
38
liftOver VCF/BED
Filter VCF for master variants list
For each variant in master variants list:
Does call exist in filtered VCF?
Yes
No
Is it passed?
Yes
No
Genotype (GT)
0/0
1/1
else
Rule 1: pass
Calls list as ref
Rule 1: pass
Calls list as alt
Rule 3: rejected
Calls list as reject
Process all calls above limit
Is call covered in BED?
Yes
No (no call)
Rule 2: covered, no call
Calls list as ref
NEXT
RETURN



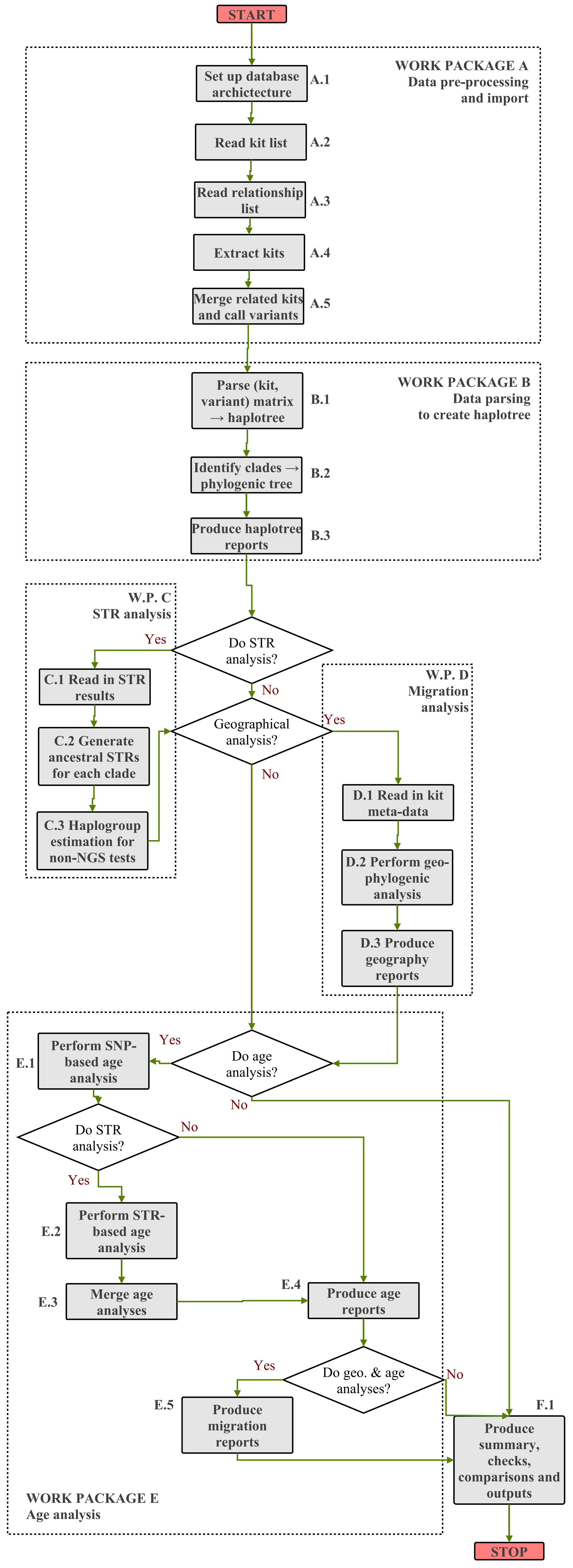
START
WORK PACKAGE A Data pre-processing and import
Set up database architecture
A.1
Read kit list
A.2
Read relationship list
A.3
Extract kits
A.4
Merge related kits and call variants
A.5
WORK PACKAGE B Data parsing to create haplotree
Parse (kit, variant) matrix → haplotree
B.1
Identify clades → phylogenic tree
B.2
Produce haplotree reports
B.3
W.P. C STR analysis
Do STR analysis?
Yes
No
C.1 Read in STR results
C.2 Generate ancestral STRs for each clade
C.3 Haplogroup estimation for non-NGS tests
Geographical analysis?
Yes
No
W.P. D Migration analysis
D.1 Read in kit meta-data
D.2 Perform geo-phylogenic analysis
D.3 Produce geography reports
Do age analysis?
Yes
No
E.1
Perform SNP-based age analysis
Do STR analysis?
Yes
No
E.2
Perform STR-based age analysis
E.3
Merge age analyses
E.4
Produce age reports
Do geo. & age analyses?
Yes
No
E.5
Produce migration reports
F.1
Produce summary, checks, comparisons and outputs
WORK PACKAGE E Age analysis
STOP

Item B.1: Parse variants to form a haplotree

This is one of the more computationally and logically challenging aspects of this process. There are several issues:

- Although rare, a handful of mutations may back-mutate in individual lines, and we need to correct for this.
- Mutations can be recurrent, meaning they cannot be parsed into a phylogenic tree, and need split up.
- Many mutations will be called sporadically among a few tests.
- Indel notation needs standardised among the tests: https://genome.sph.umich.edu/wiki/Variant_Normalization

Many of these processes have known solutions. However, the common solutions, e.g., http://www.it2kane.org/2016/07/combining-variant-compare-files/ have problems with scalability, reference positives, treating bad data and questionable variants. A more streamlined approach is desired that reduces both CPU time and RAM.


START
WORK PACKAGE A
Data pre-processing and import
Set up database archictecture
A.1
Read kit list
A.2
Read relationship list
A.3
Extract kits
A.4
Merge related kits and call variants
A.5
WORK PACKAGE B
Data parsing to create haplotree
Parse (kit, variant) matrix → haplotree
B.1
Identify clades → phylogenic tree
B.2
Produce haplotree reports
B.3
W.P. C
STR analysis
Do STR analysis?
Yes
No
C.1 Read in STR results
C.2 Generate ancestral STRs for each clade
C.3 Haplogroup estimation for non-NGS tests
Geographical analysis?
Yes
No
W.P. D
Migration analysis
D.1 Read in kit meta-data
D.2 Perform geo-phylogenic analysis
D.3 Produce geography reports
Do age analysis?
Yes
No
E.1
Perform SNP-based age analysis
Do STR analysis?
No
Yes
E.2
Perform STR-based age analysis
E.3
Merge age analyses
E.4
Produce age reports
Do geo. & age analyses?
Yes
No
E.5
Produce migration reports
F.1
Produce summary, checks, comparisons and outputs
WORK PACKAGE E
Age analysis
STOP

Extended haplotree example

Names are for illustrative purposes only, and not used in the logic

Matrix of {people,variants}

hg38	Name	A	B	C	D	E	F	G	H	I	J
3019783	M343	+	+	?	+	+	+	+	+	+	+
6920349	Z9	-	-	-	+	-	-	-	-	+	-
7378685	Z381	-	+	+	?	-	-	+	+	+	-
8928037	U106	+	+	+	+	-	+	+	+	+	-
12060401	Z301	?	?	+	?	?	-	?	+	?	?
12879820	Z18	+	-	-	-	-	+	-	-	-	-
13668461	Z156	-	+	-	-	-	-	+	-	-	-
15732138	L11	+	+	+	+	+	?	+	+	+	+
19538924	Z28	-	-	-	+	-	?	-	-	+	-
19995425	P312	-	-	-	-	+	-	-	-	-	+
20029258	Z8	-	-	-	+	-	-	-	-	-	-
20323911	A297	-	?	-	+	-	-	+	-	?	-
20577481	M269	+	+	+	+	+	+	+	+	+	+
20625892	Z306	-	+	-	-	-	-	-	-	-	-
21450311	L48	-	-	+	+	-	-	-	+	+	-

Split mutations never called negative
Split perfect/imperfect calls
Sort (im)perfects both by # positives,
then by first positive column

hg38	Name	A	B	C	D	E	F	G	H	I	J
3019783	M343	+	+	?	+	+	+	+	+	+	+
15732138	L11	+	+	+	+	+	?	+	+	+	+
20577481	M269	+	+	+	+	+	+	+	+	+	+
8928037	U106	+	+	+	+	-	+	+	+	+	-
21450311	L48	-	-	+	+	-	-	-	+	+	-
6920349	Z9	-	-	-	+	-	-	-	-	+	-
12879820	Z18	+	-	-	-	-	+	-	-	-	-
13668461	Z156	-	+	-	-	-	-	+	-	-	-
19995425	P312	-	-	-	-	+	-	-	-	-	+
20029258	Z8	-	-	-	+	-	-	-	-	-	-
20625892	Z306	-	+	-	-	-	-	-	-	-	-
7378685	Z381	-	+	+	?	-	-	+	+	+	-
12060401	Z301	?	?	+	?	?	-	?	+	?	?
19538924	Z28	-	-	-	+	-	?	-	-	+	-
20323911	A297	-	?	-	+	-	-	+	-	?	-

Horizontal sort perfect calls to create clades

hg38	Name	D	I	C	H	A	F	B	G	E	J
8928037	U106	+	+	+	+	+	+	+	+	-	-
21450311	L48	+	+	+	+	-	-	-	-	-	-
6920349	Z9	+	+	-	-	-	-	-	-	-	-
12879820	Z18	-	-	-	-	+	+	-	-	-	-
13668461	Z156	-	-	-	-	-	-	+	+	-	-
19995425	P312	-	-	-	-	-	-	-	-	+	+
20029258	Z8	+	-	-	-	-	-	-	-	-	-
20625892	Z306	-	-	-	-	-	-	+	-	-	-
7378685	Z381	?	+	+	+	-	-	+	+	-	-
12060401	Z301	?	?	+	+	?	-	?	?	?	?
19538924	Z28	+	+	-	-	-	?	-	-	-	-
20323911	A297	+	?	-	-	-	-	?	+	-	-

Select imperfects where one call is missing and fit in from top→bottom:
Z381 logic:
Z381 has 5-6 +ves → between L48 and U106
Work from U106 to L48:
Test U106 for positives: I,C,H,B,G are U106+ Z381+
Test Z381 for U106: A,F are U106+ Z381- so Z381 is a subset of U106
Test Z381 for U106: no U106- Z381+ so phylogenically consistent
Test L48 for positives: C,H are L48+ Z381+
Test L48 for U106: B,G are L48- Z381+ so Z381 is a superset of L48&?
D is L48+ so D must be Z381+
All calls fixed, insert into tree

Z28 logic:
Z28 has 2-3 +ves → P312 (lowest 2x) and L48
Work from L48 to P312:
Test L48 for positives: D,I are L48+
Test Z28 for L48: C,H are L48+ but Z28-, so Z28 is a subset of L48
Test Z28 for L48: no L48- Z28+, so phylogenically consistent
Test Z9 for positives: D,I are Z9+
Test Z28 for Z9: no Z9+ Z28-, so Z28 may be equivalent to Z9
Test Z28 for Z9: no Z9- Z28+, so Z28 is consistent and equivalent to Z9
Test Z18: Z18 is L48-, and A is Z28-, so F must be Z28-
All calls fixed, insert into tree

Horizontal sort new matrix to create clades
(Swaps AF and BG)
Vertical sort swaps Z18 and Z156
(can be needed to merge SNPs in a clade)

hg38	Name	D	I	C	H	B	G	A	F	E	J
8928037	U106	+	+	+	+	+	+	+	+	-	-
7378685	Z381	+	+	+	+	+	+	-	-	-	-
21450311	L48	+	+	+	+	-	-	-	-	-	-
6920349	Z9	+	+	-	-	-	-	-	-	-	-
19538924	Z28	+	+	-	-	-	-	-	-	-	-
13668461	Z156	-	-	-	-	+	+	-	-	-	-
12879820	Z18	-	-	-	-	-	-	+	+	-	-
19995425	P312	-	-	-	-	-	-	-	-	+	+
20029258	Z8	+	-	-	-	-	-	-	-	-	-
20625892	Z306	-	-	-	-	+	-	-	-	-	-
12060401	Z301	?	?	+	+	?	?	?	-	?	?
20323911	A297	+	?	-	-	?	+	-	-	-	-

Select imperfects where two calls are missing and fit in from top→bottom:
A297 logic:
A297 has 2-4 +ves → between P312 and L48
Work from L48 to P312:
Test L48 for positives: D and G are positives
Test A297 for L48: C,H are L48+ A297- so A297 is a subset of L48
Test A297 for L48: G is L48- A297+ so A297 is phylogenically inconsistent
→ Recurrent or junk - do not insert yet
Horizontal sort, vertical sort

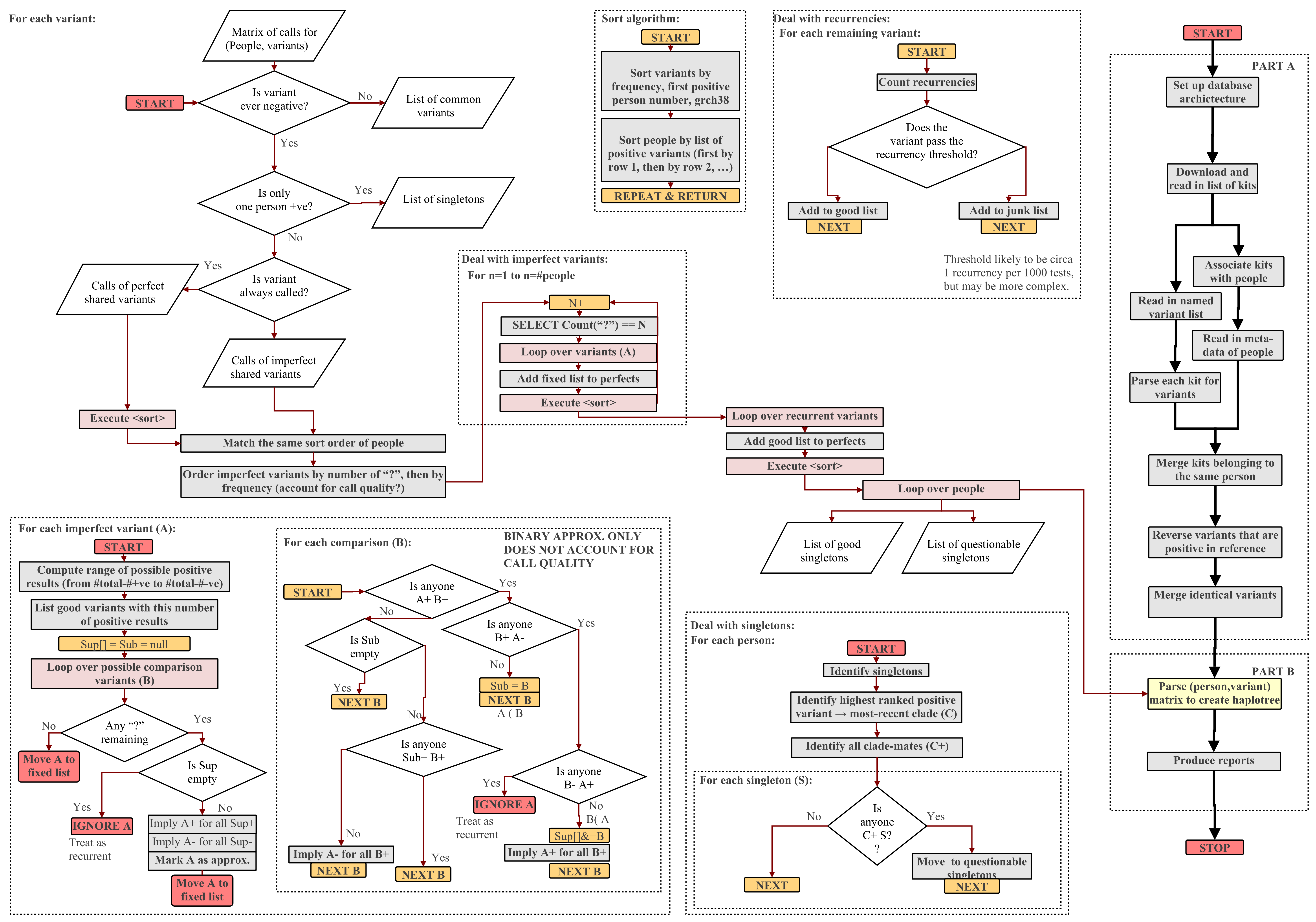
…Select imperfects where seven calls are missing and fit in from top→bottom:
Z301 logic:
Z301 has 2-9 +ves → between P312 and U106
Work from U106 to P312:
Test U106 for positives: C,H are U106+ Z301+
Test Z301 for U106: F is U106+ Z301- so Z301 is a subset of U106
Test Z301 for U106: no U106- Z301+ so phylogenically consistent
Test Z381 for positives: C,H are Z381+ Z301+
Test Z301 for Z381: no Z381+ Z301- so Z301 may be equivalent to Z381
Test Z301 for Z381: no Z381- Z301+ so Z301 is consistent and equivalent to Z381
Test L48 for positives: C,H are L48+ Z301+
Test Z301 for L48: no L48+ Z301- so Z301 may be equivalent to L48
Test Z301 for L48: no L48- Z301+ so Z301 is consistent and equivalent to L48
Test Z9: no Z9+ Z301+ so Z301 is not a superset of Z9; Z9 is also a subset of U106
Test Z28: no Z28+ Z301+ so Z301 is not a superset of Z28; Z28 is also a subset of U106
Test Z156: no Z156+ Z301+ so Z301 is not a superset of Z156; Z156 is also a subset of U106
Test Z18: no Z18+ Z301+ so Z301 is not a superset of Z18; ; Z18 is also a subset of U106
Test P312: no P312+ Z301+ so Z301 is not a superset of P312
P312 is not a subset of U106, so E,J are Z301-
Z301 is equivalent to U106>Z381 or U106>Z381>L48
Test bottom up: L48, Z381:
D,I are L48+ so are Z301+
B,G are Z381+ so may be Z301+ - presume positive in tree, merge up
Ambiguous calls remaining, insert into tree at highest possible level
Flag as ambiguous position

Return to recurrent/junk mutations:
(Call mutations with two recurrencies first, proceed from best to worst called)
A297 logic:
Two recurrencies in ten tests
Does this exceed a maximum tolerance?
Maximum tolerance would normally be ~1 in 1000 so this would be junk, but let's say no…
Split recurrencies: A297.1, A297.2, parse as before:
→ A297.1 is either equivalent to Z8 or Z9/Z28, parse as a questionable singleton
→ A297.2 is either equivalent to Z306 or Z156, parse as a questionable singleton
Questionable singletons don't get merged up, so we assume A297.1=Z8, A297.2=Z306
Horizontal sort, vertical sort
All mutations assigned → Haplotree complete,

hg38	Name	D	I	C	H	B	G	A	F	E	J
8928037	U106	+	+	+	+	+	+	+	+	-	-
7378685	Z381	+	+	+	+	+	+	-	-	-	-
12060401	Z301	+	+	+	+	+	+	-	-	-	-
21450311	L48	+	+	+	+	-	-	-	-	-	-
6920349	Z9	+	+	-	-	-	-	-	-	-	-
19538924	Z28	+	+	-	-	-	-	-	-	-	-
13668461	Z156	-	-	-	-	+	+	-	-	-	-
12879820	Z18	-	-	-	-	-	-	+	+	-	-
19995425	P312	-	-	-	-	-	-	-	-	+	+
20029258	Z8	+	-	-	-	-	-	-	-	-	-
20323911	A297.1	+	-	-	-	-	-	-	-	-	-
20625892	Z306	-	-	-	-	+	-	-	-	-	-
20323911	A297.2	-	-	-	-	-	+	-	-	-	-

form clades

D	I	C	H	B	G	A	F	E	J
+	+	+	+	+	+	+	+	-	-
+	+	+	+	+	+	-	-	-	-
+	+	+	+	+	+	-	-	-	-
+	+	+	+	-	-	-	-	-	-
+	+	-	-	-	-	-	-	-	-
+	+	-	-	-	-	-	-	-	-
-	-	-	-	+	+	-	-	-	-
-	-	-	-	-	-	+	+	-	-
-	-	-	-	-	-	-	-	+	+
+	-	-	-	-	-	-	-	-	-
+	-	-	-	-	-	-	-	-	-
-	-	-	-	+	-	-	-	-	-
-	-	-	-	-	+	-	-	-	-


For each variant:
Matrix of calls for (People, variants)
START
Is variant ever negative?
No
List of common variants
Yes
Is only one person +ve?
Yes
List of singletons
No
Is variant always called?
Yes
Calls of perfect shared variants
Calls of imperfect shared variants
Execute <sort>
Match the same sort order of people
Order imperfect variants by number of "?", then by frequency (account for call quality?)
Sort algorithm:
START
Sort variants by frequency, first positive person number, grch38
Sort people by list of positive variants (first by row 1, then by row 2, …)
REPEAT & RETURN
Deal with recurrencies:
For each remaining variant:
START
Count recurrencies
Does the variant pass the recurrency threshold?
Add to good list
NEXT
Add to junk list
NEXT
Threshold likely to be circa 1 recurrency per 1000 tests, but may be more complex.
Deal with imperfect variants:
For n=1 to n=#people
N++
SELECT Count("?") == N
Loop over variants (A)
Add fixed list to perfects
Execute <sort>
Loop over recurrent variants
Add good list to perfects
Execute <sort>
Loop over people
List of good singletons
List of questionable singletons
For each imperfect variant (A):
START
Compute range of possible positive results (from #total-#+ve to #total-#-ve)
List good variants with this number of positive results
Sup[] = Sub = null
Loop over possible comparison variants (B)
Any "?" remaining
No
Move A to fixed list
Yes
Is Sup empty
Yes
IGNORE A
Treat as recurrent
No
Imply A+ for all Sup+
Imply A- for all Sup-
Mark A as approx.
Move A to fixed list
For each comparison (B):
BINARY APPROX. ONLY DOES NOT ACCOUNT FOR CALL QUALITY
START
Is anyone A+ B+
Yes
No
Is Sub empty
Yes
NEXT B
No
Is anyone B+ A-
Yes
No
Sub = B
NEXT B
A (B
Is anyone Sub+ B+
No
Imply A- for all B+
NEXT B
Yes
NEXT B
Is anyone B- A+
Yes
IGNORE A
Treat as recurrent
No
B(A
Sup[]&=B
Imply A+ for all B+
NEXT B
Deal with singletons:
For each person:
START
Identify singletons
Identify highest ranked positive variant → most-recent clade (C)
Identify all clade-mates (C+)
For each singleton (S):
Is anyone C+ S? ?
No
NEXT
Yes
Move to questionable singletons
NEXT
START
PART A
Set up database archictecture
Download and read in list of kits
Associate kits with people
Read in named variant list
Read in meta-data of people
Parse each kit for variants
Merge kits belonging to the same person
Reverse variants that are positive in reference
Merge identical variants
PART B
Parse (person,variant) matrix to create haplotree
Produce reports
STOP