

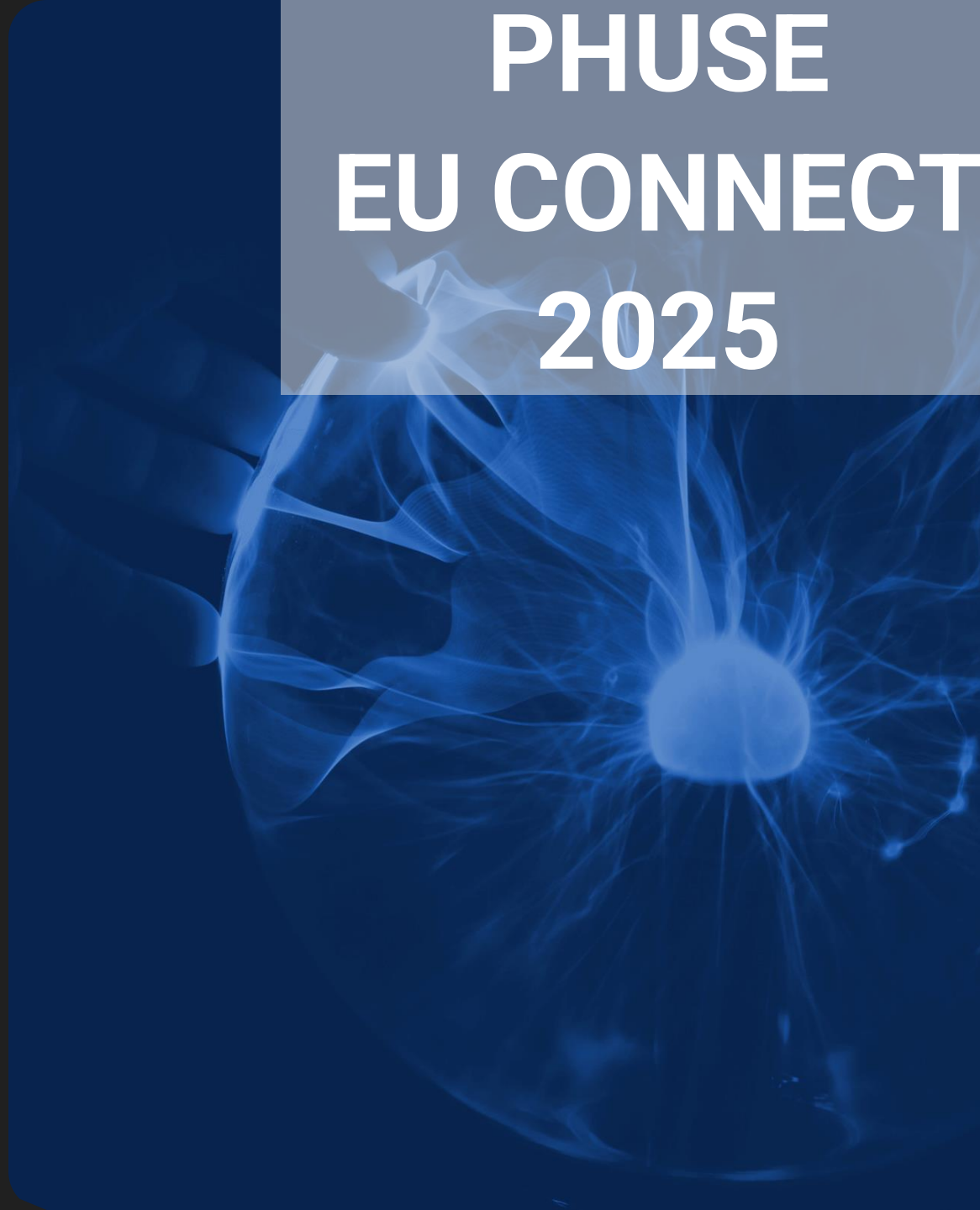
SM03: Untangling the Dependencies Web

Rhys McCrosson & Simon Purkess

Tuesday 18th November 2025



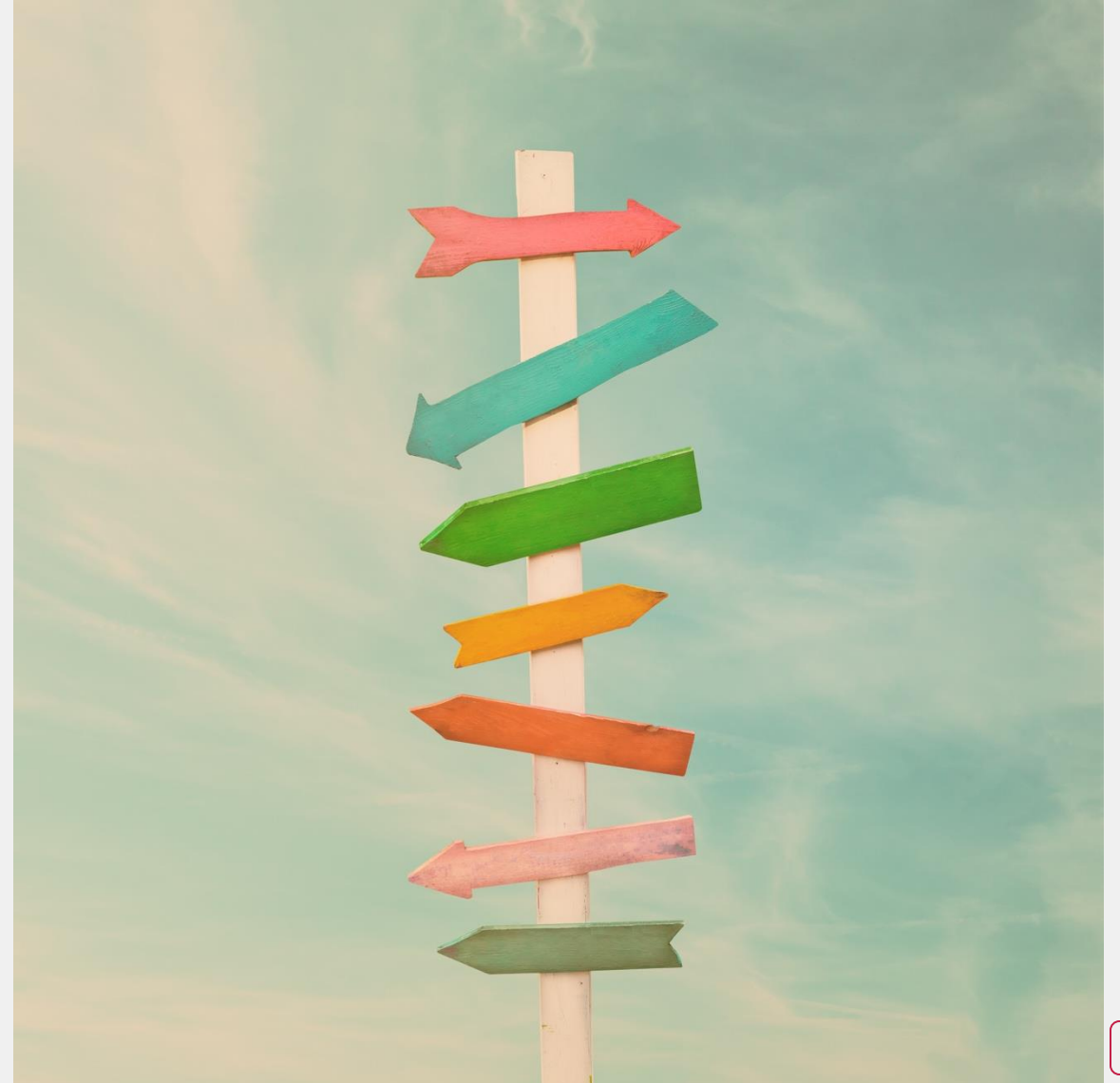
PHUSE
EU CONNECT
2025



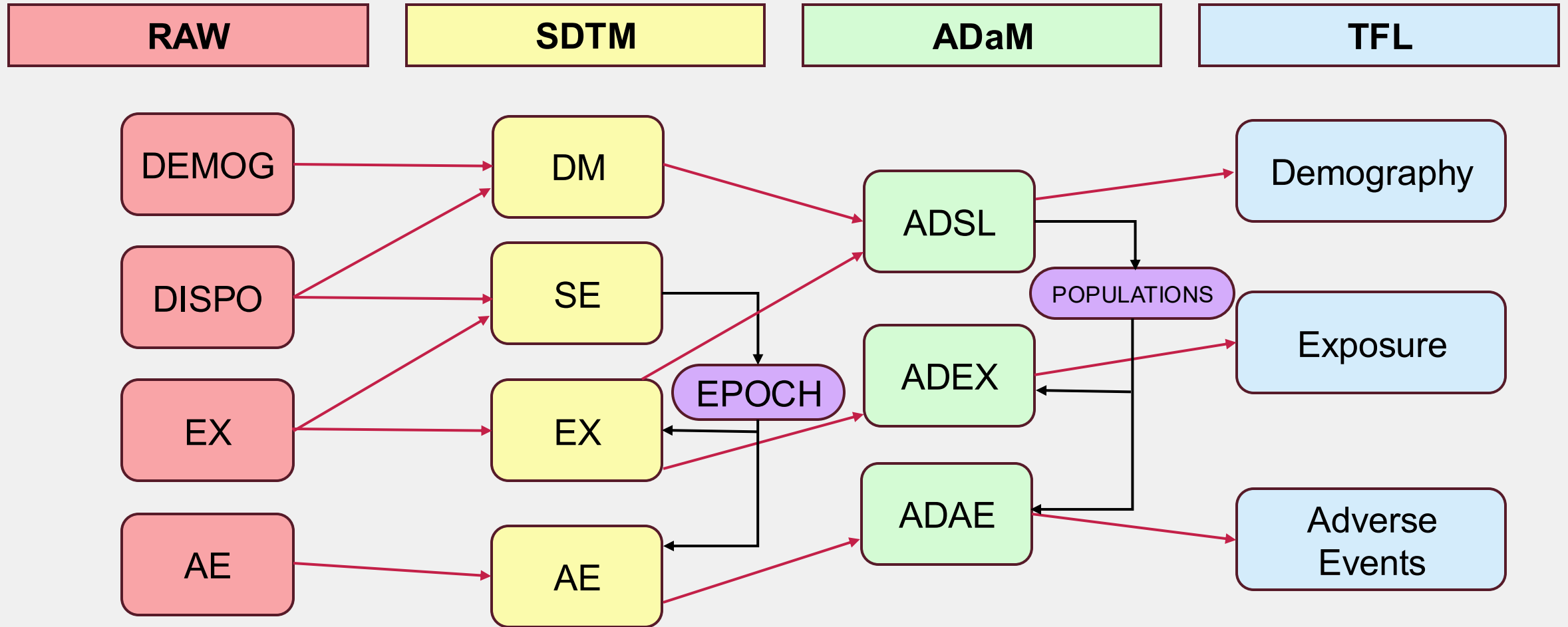
What are dependencies?

Any situation where programs must be run in a specific order

- Cross-level vs Within-level
- Direct reference vs Indirect (macro)



The Dependencies Web

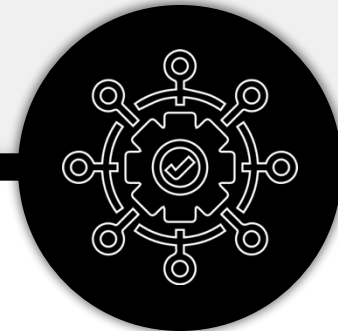


How to identify dependencies?



Manual Review

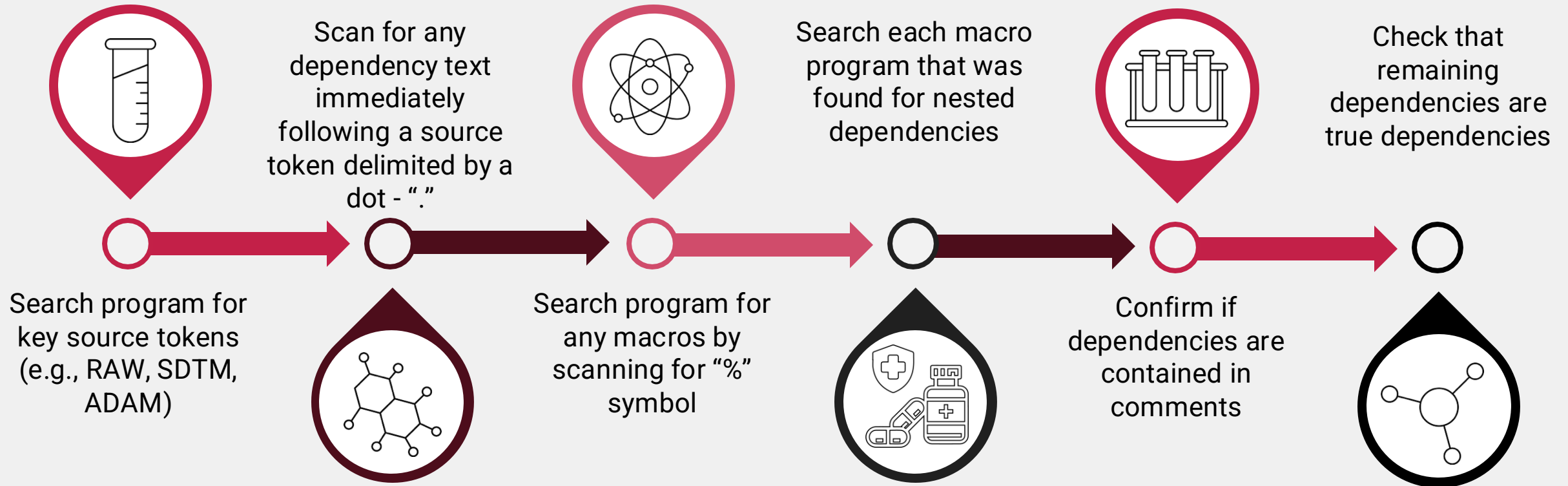
- Program headers
- QC Trackers
- Dataset Specifications



Text-based Search

- A better way?

Text-based search approach (simplified)



True vs False dependencies

True Dependencies

- True dependencies are identified from input data statements from non-commented code.

False Dependencies

- False dependencies are contained within commented code or can be identified from output data statements.

```
*input ADAM.ADAE data;
data adae;
  set adae.adae;
run;

*input ADAM.ADSL data;
data adsl;
  set adam.adsl;
/*  set adam.adslx; */
run;

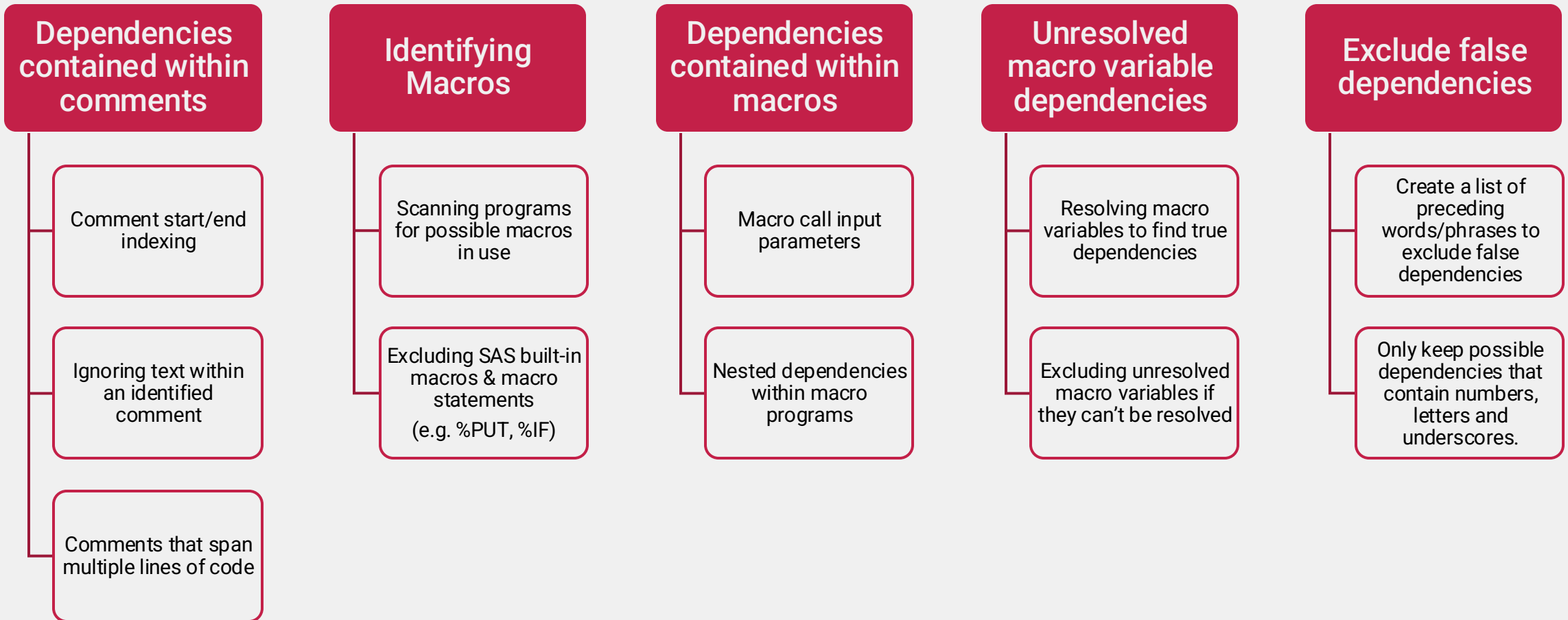
*sort data for merge;
proc sort data = adae;
  by usubjid;
run;

proc sort data = adsl;
  by usubjid;
run;

*merge adam.adae and adam.adsl;
data merged;
  merge adae(in=a) adsl(in=b);
  by usubjid;
run;

*output data;
proc sort data = merged out = output.ae201;
  by usubjid paramn astdt;
run;
```

Text-based search - challenges



Finding dependencies contained in comments – example code

Find comment start/end points if they exist.

Comment Start	Comment End
*	;
/*	*/
%*	;

Check if an identified source exists within the comment start/end points.

```
/* Filter out dependencies contained within comments for specified sources */  
if (0 < comment_start < index(uppercase(line), 'ADAM.') < comment_end)  
    or (0 < comment_start < index(uppercase(line), 'SDTM.') < comment_end)  
    or (0 < comment_start < index(uppercase(line), 'RAW.') < comment_end)  
then comment = 1;  
else comment = 0;  
end;  
else comment = 0;
```

Finding dependencies contained within macro calls – example code

Find start/end points for parameter calls using parenthesis and equal sign

Use macro start/end points to obtain all parameters from macro calls

```
if index(uppercase(line), '%COMBINESDTM') > 0 or index(uppercase(line), '%DSIN') > 0 then do;

  open  = index(line, '(');
  equal = index(line, '=');
  close = index(line, ')');

  if open > 0 and close > 0 then do;
    if equal > 0 and open < equal and equal < close then do;
      start = equal+1;
      end   = close-equal-1;
    end;
    else if open < close then do;
      start = open+1;
      end   = close-open-1;
    end;
    else do;
      call missing(temp_dep);
      start = .;
      end   = .;
    end;

    if start > 0 and end > 0 then temp_dep = strip(substr(line, start, end));
    else call missing(temp_dep);
  end;
  else call missing(temp_dep);
```

Unresolved macro variables



Unresolved macro variables could sometimes be identified as dependencies.



These macro variables are not incorrect although they may already be accounted for by other found dependencies.



Unresolved macro variables do not provide useful information as dependencies.

program	dependencies
AE202	ADAM.ADAE, ADAM.ADSL
AE202A	ADAM.ADAE, ADAM.ADSL
AE202D	ADAM.ADAE, ADAM.ADMH, ADAM.ADSL, OUTPUT.&PROGNAME
AE204	ADAM.ADAE, ADAM.ADSL
AE206	ADAM.ADAE, ADAM.ADSL, SANDBOX.QC_AE206
AE206A	ADAM.ADAE, ADAM.ADSL
AE208	ADAM.ADAE, ADAM.ADSL
AE208A	ADAM.ADAE, ADAM.ADSL
AE209	ADAM.ADAE, ADAM.ADSL
AE209A	ADAM.ADAE, ADAM.ADSL
AE212	ADAM.ADAE, ADAM.ADSL
AE212A	ADAM.ADAE, ADAM.ADSL
AE212AJ	ADAM.ADAE, ADAM.ADSL

Unresolved macro variables – solutions?

Method	Comment
Macro variables could be stored globally for a given program.	This is unlikely for most programs.
Programs could be executed in real-time to obtain a list of locally stored macro variables and their resolved forms.	We would not want to update or execute programs that we are scanning for dependencies.
Program logs could be investigated for the macro variables and their resolved forms.	This could only work if programs contained the relevant options to always show resolved macro variables.
Remove macro variable dependencies from the list of dependencies.	This is feasible although can potentially lead to missing out valid dependencies.

Predecessor dependency tokens

Dependency Validation

- We need to know if a possible dependency comes from an input or output statement in a program.
- We can figure this out by checking for key text phrases immediately before a dependency was identified.

Exclusion List

- A list of exclusion terms must be created., e.g. "CREATE TABLE", "OUT"
- Dependencies immediately following an exclusion term can should be considered as false dependencies.
- In contrast, preceding terms like "IN=" or "SET" would indicate a true dependency.

How to build the Exclusion List?

1. Create a list of possible terms to exclude.
2. Loop around each exclusion term.
3. Calculate the length of each exclusion term to know how many preceding characters to check before a dependency is identified.
4. If the preceding text matches the exclusion term, then do not record the dependency.

```

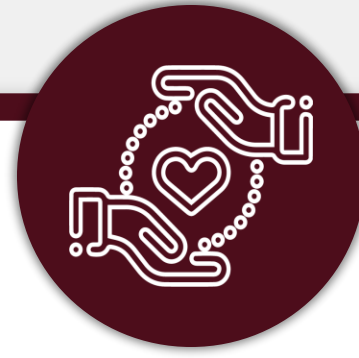
/* Create an exclusion list of preceding terms to exclude fake dependencies */
%macro excl_clause(excl_list=);
  %global excl_clause;
  %let excl_clause =;
  %do _i = 1 %to %sysfunc(countw(&excl_list, %str( )));
    %let term = %scan(&excl_list, &i, %str( ));
    %let tlen = %length(&term);
    %let excl_clause = &excl_clause
      and not(
        pos>&tlen
        and substr(upcase(line), pos-&tlen, &tlen)= "%upcase(&term)"
      );
  %end;
%mend;
```

Cross-checking dependencies



Production & QC

Comparing Production/QC program dependencies can be useful, although differences in dependencies can occur without this being an issue



ADaM Specifications

ADaM specifications could be used to identify ADaM/SDTM dependencies for further comparison with a given ADaM.

Cross-checking dependencies



Comparing programs with the QC Tracker

Dependencies can often be recorded in the QC tracker. These dependencies are usually inputted either by checking programs themselves at some point or by stating what would generally be expected to be found in a program. It can be useful to compare program dependencies with the QC tracker for the following purposes:

Check if a program contains expected dependencies from QC tracker.

Check if any dependencies are missing from a program that were recorded in the tracker.

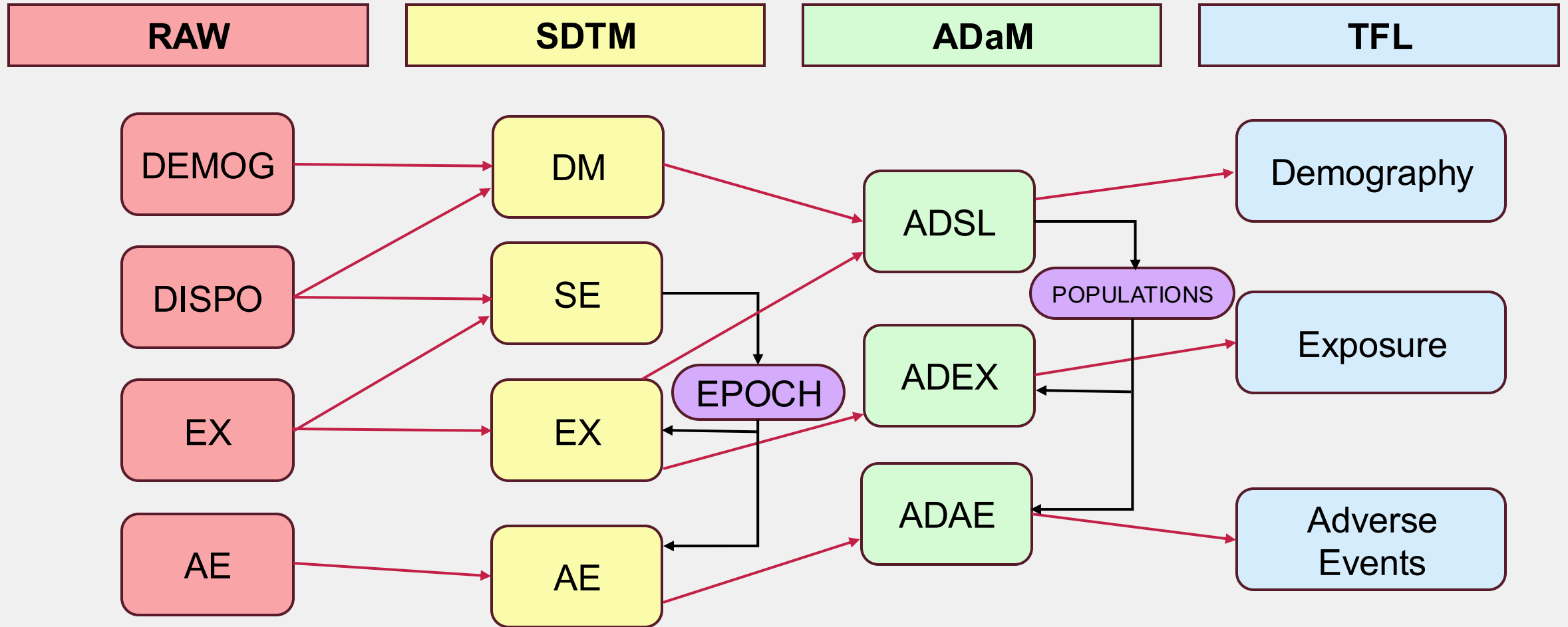
Check if the tracker is missing dependencies that exist in programs.

Check if there are any programs where no dependencies could be found in the program/qc tracker.

The Final Result

KEY RAW SDTM ADaM TFL	Dataset	Dependencies	
	DM	DEMOG	DISPO
	SE	DISPO	EX
	EX	EX	SE
	AE	AE	SE
	ADSL	DM	EX
	ADEX	ADSL	EX
	ADAE	ADSL	AE
	Demography	ADSL	
	Exposure	ADEX	
	Adverse Events	ADAE	

The Final Result



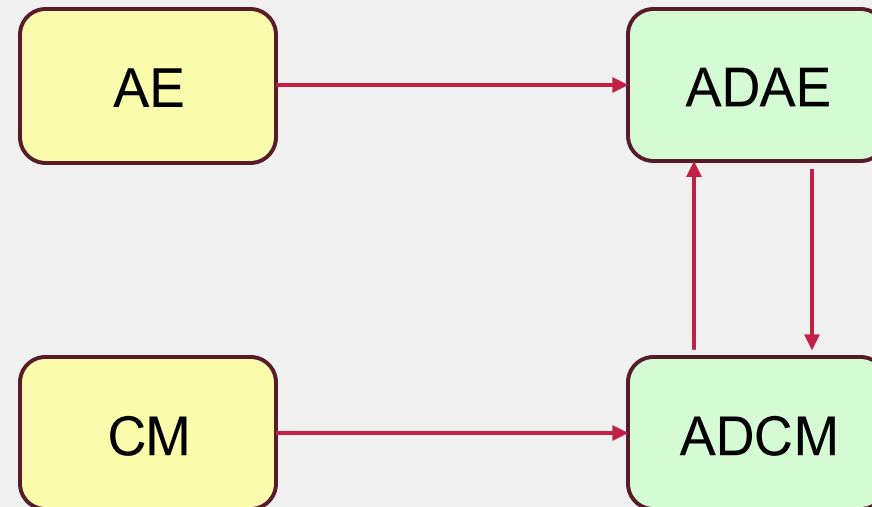
**Finding
Circular Dependencies**

Applications

Circular Dependencies

SDTM

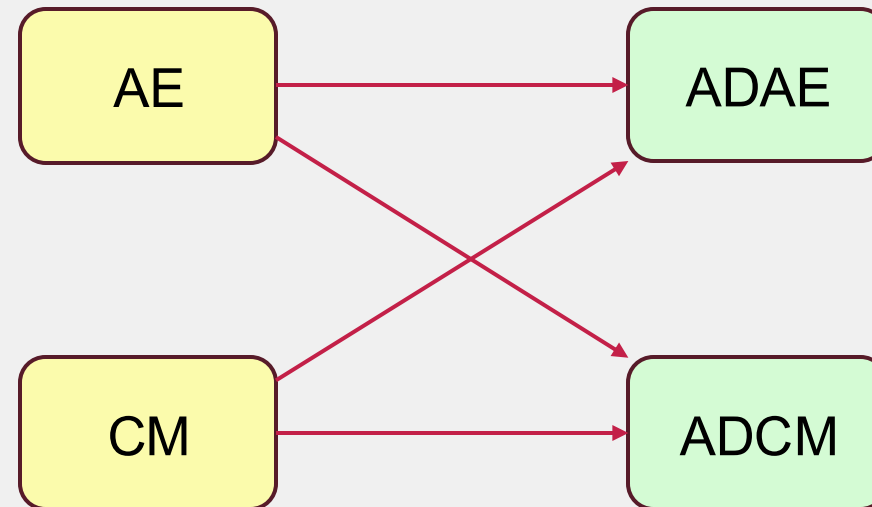
ADaM



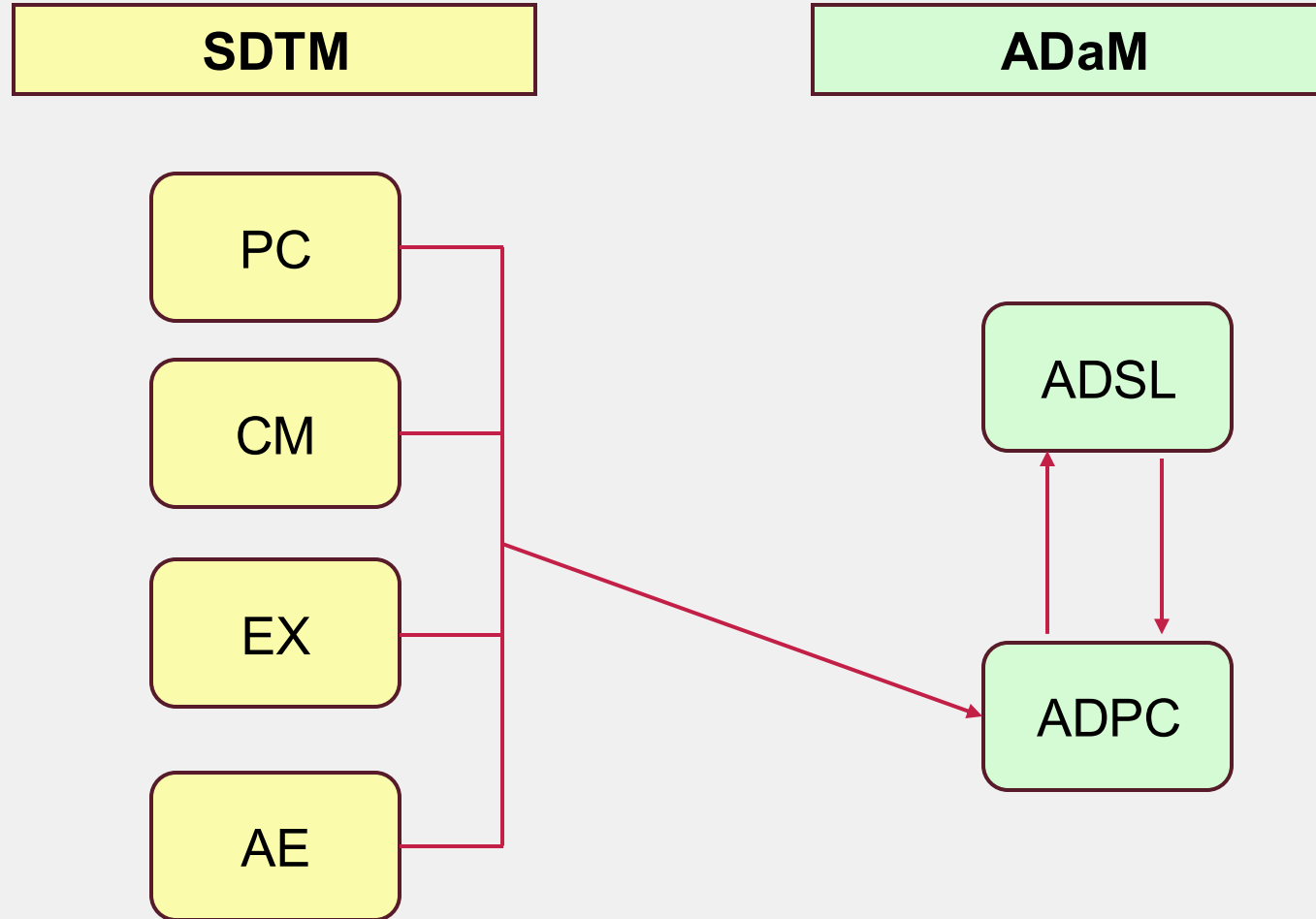
Circular Dependencies

SDTM

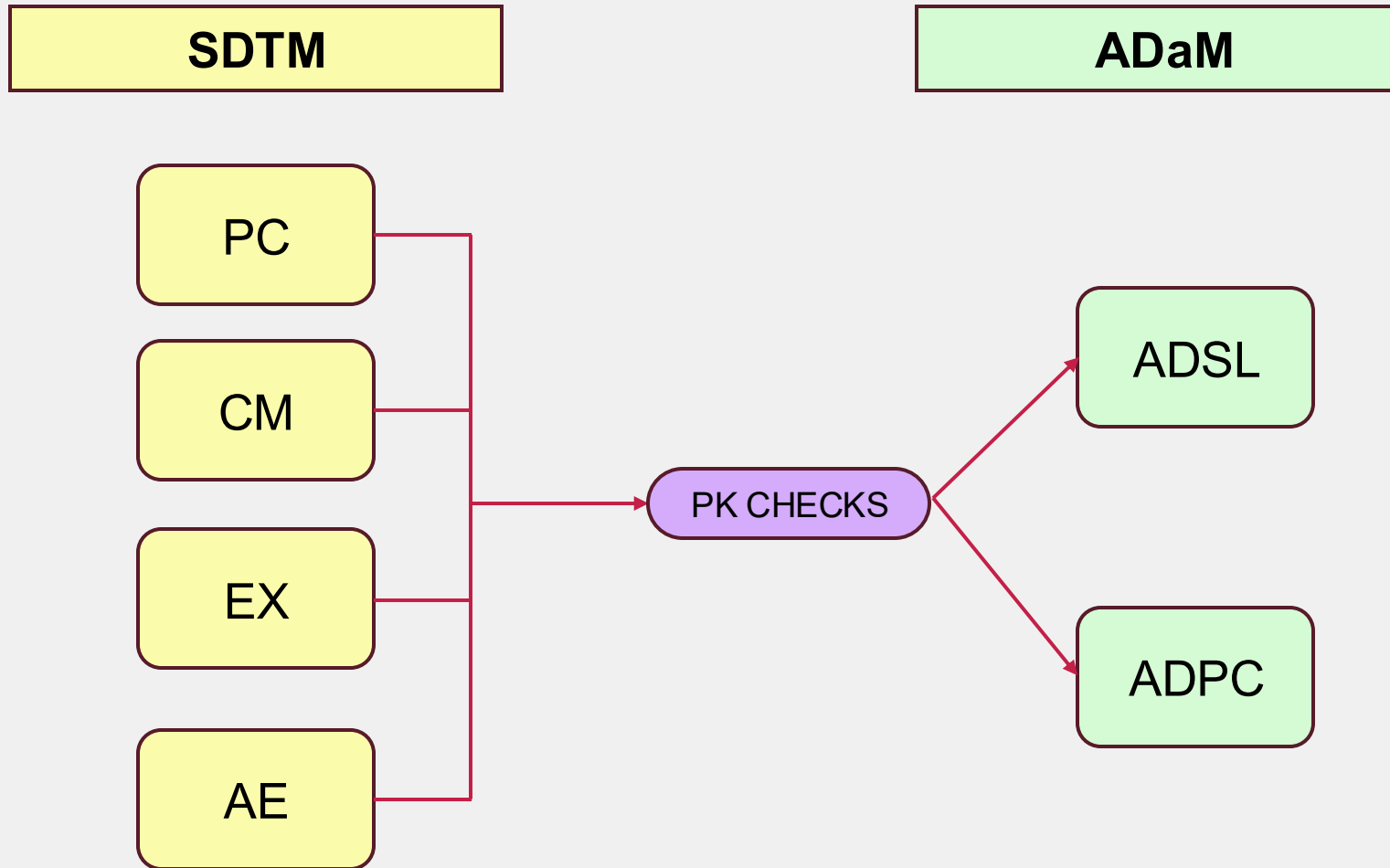
ADaM

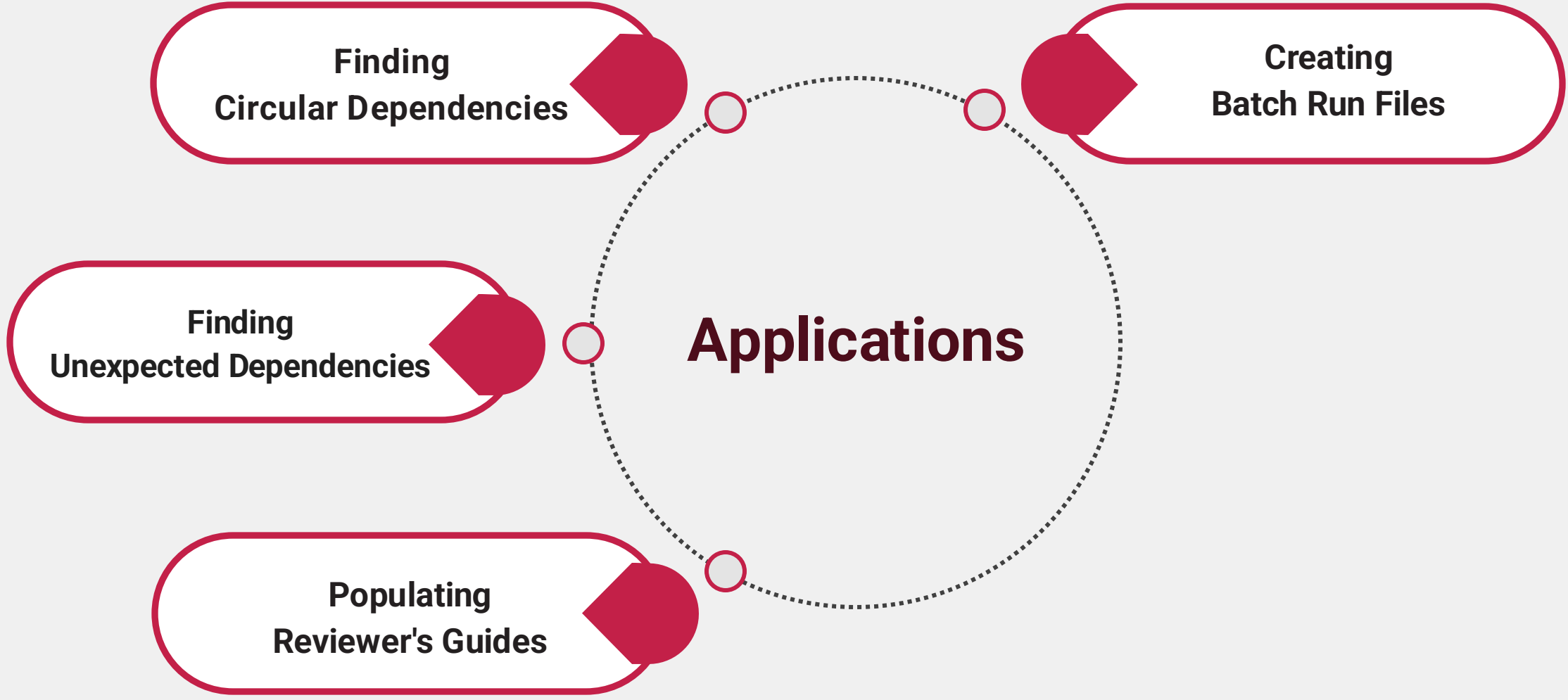


Circular Dependencies

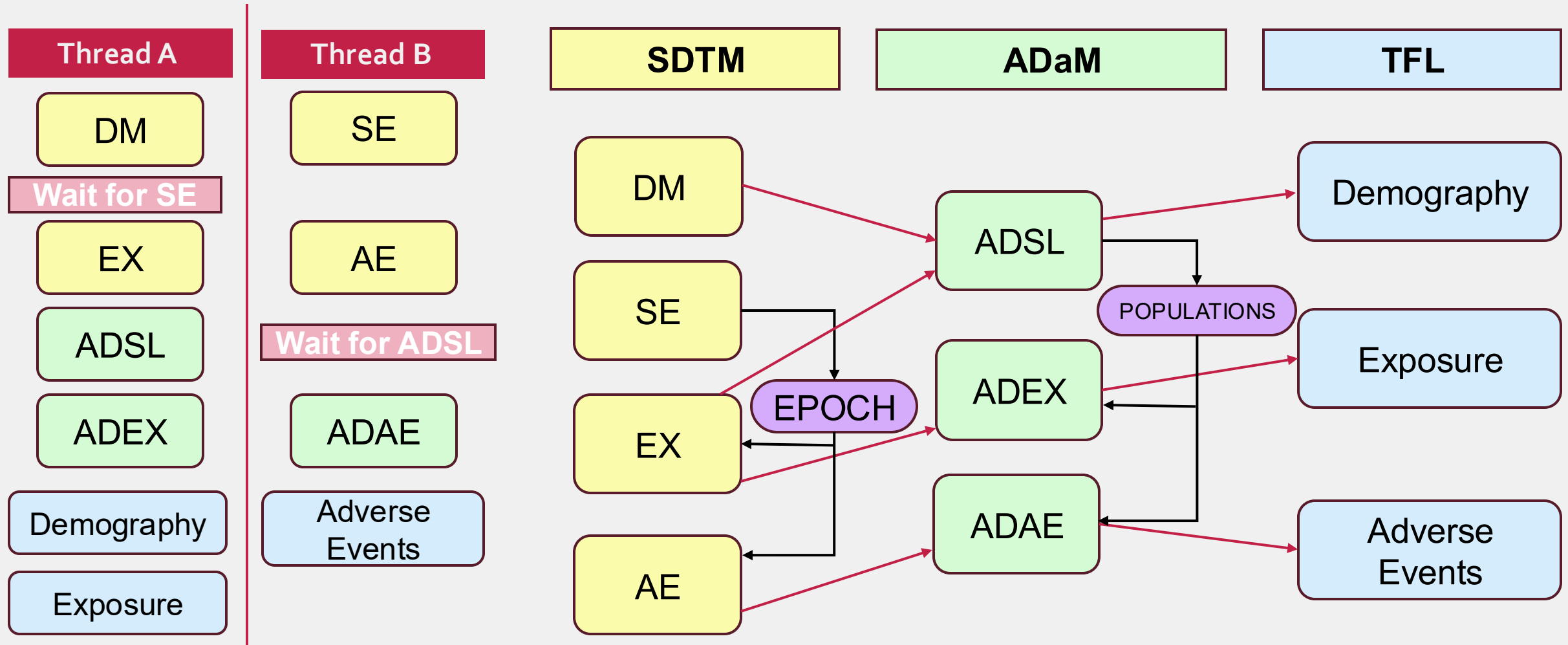


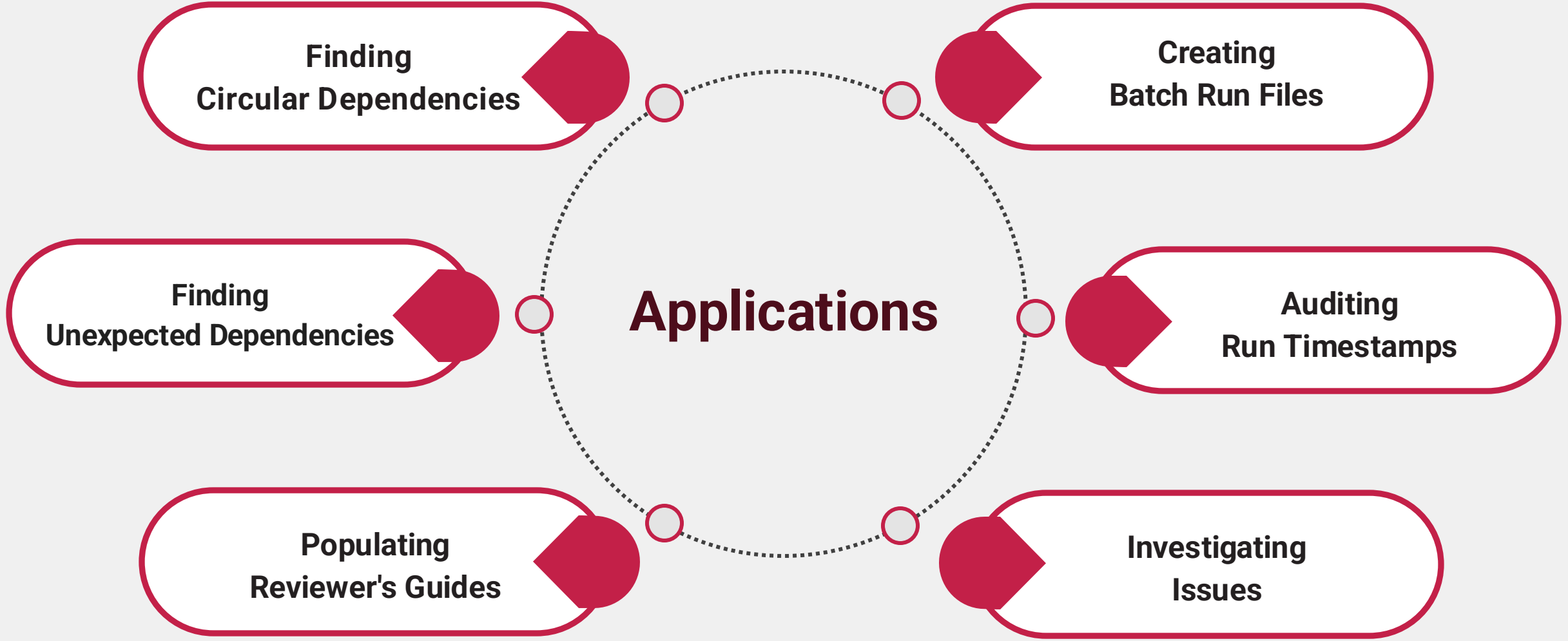
Circular Dependencies



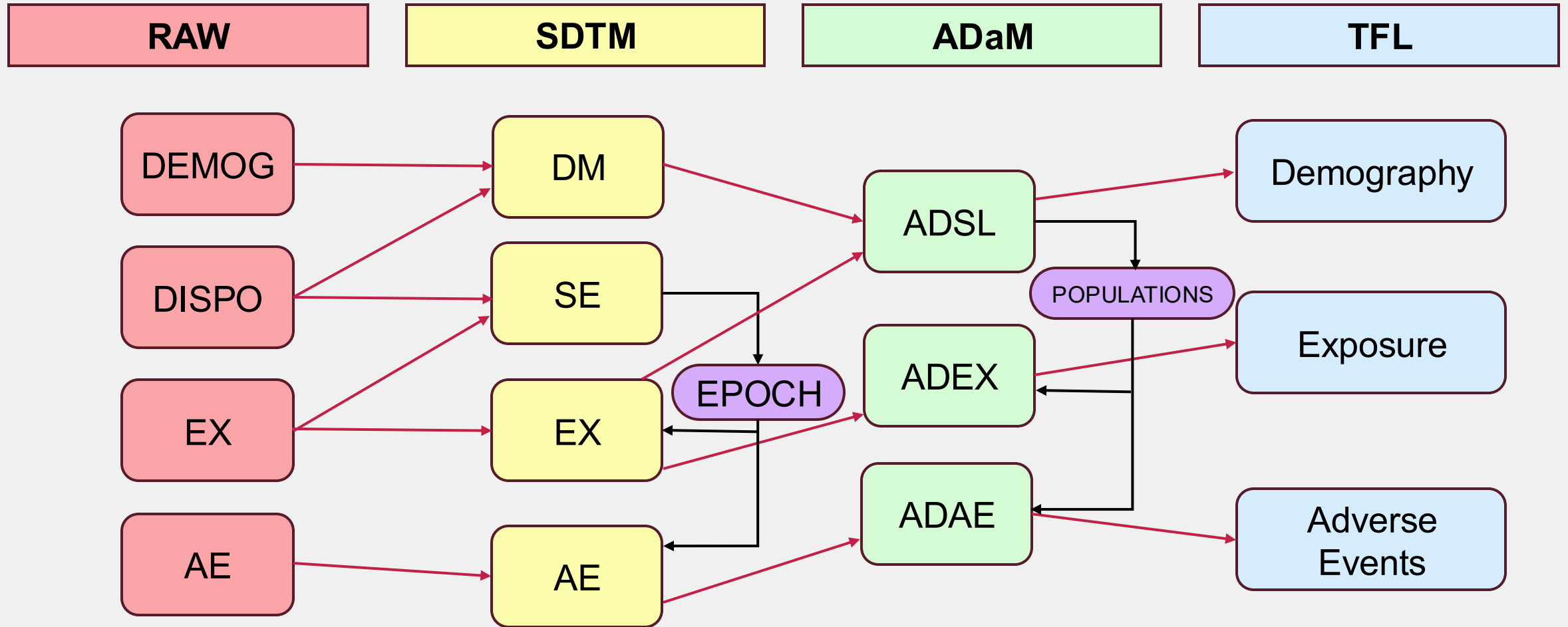


Creating a Parallel Batch File

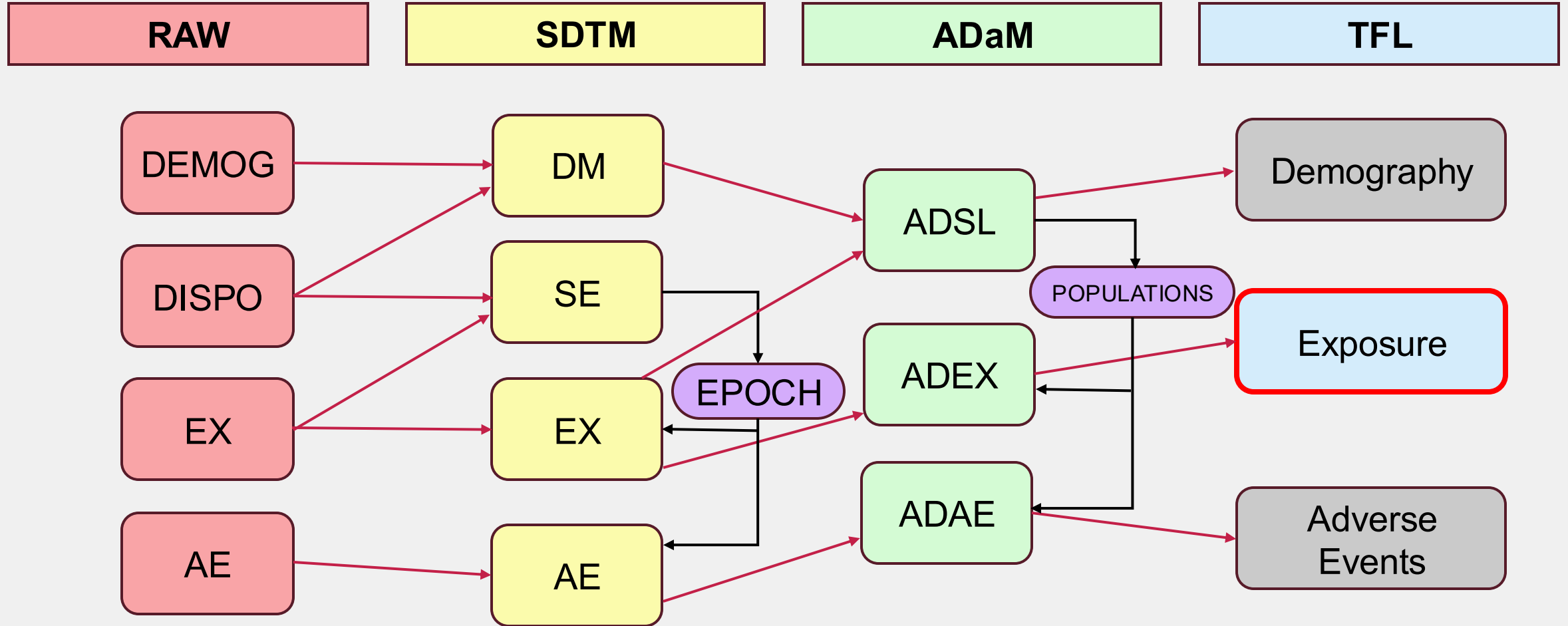




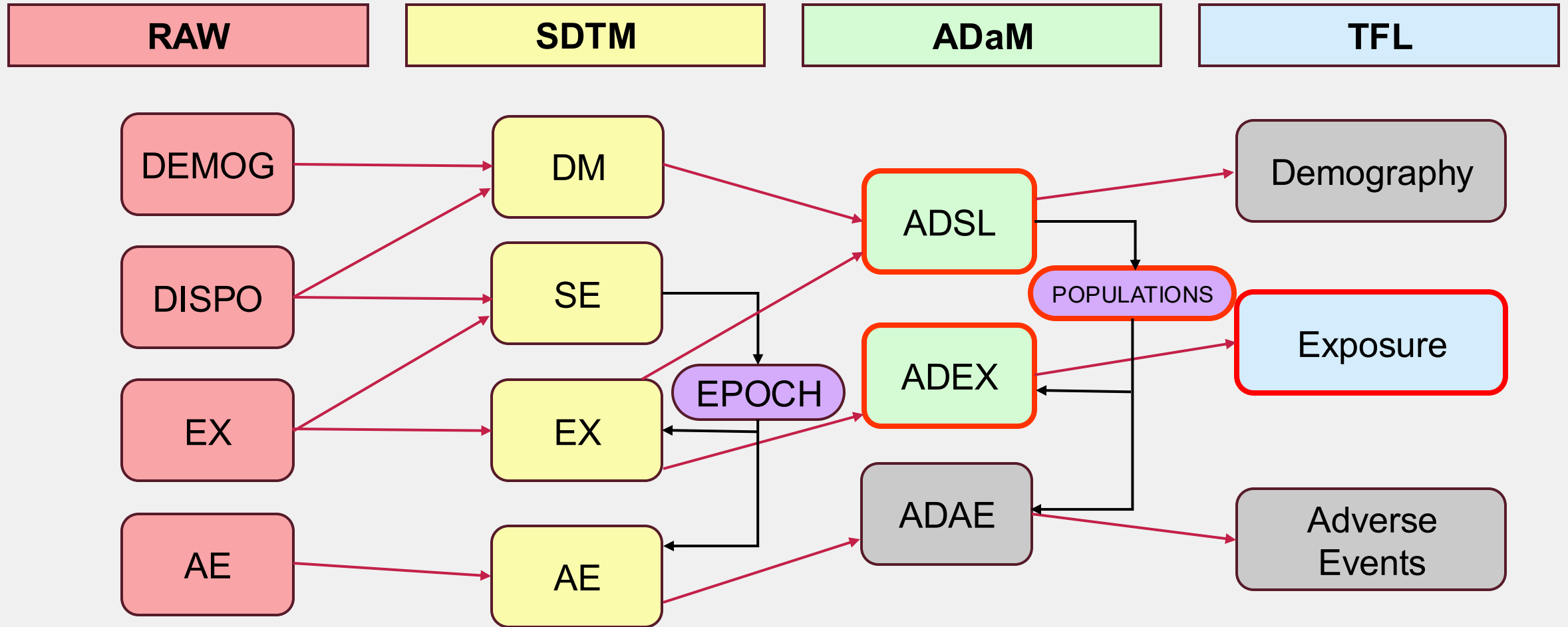
Case Study – Subset Rerun



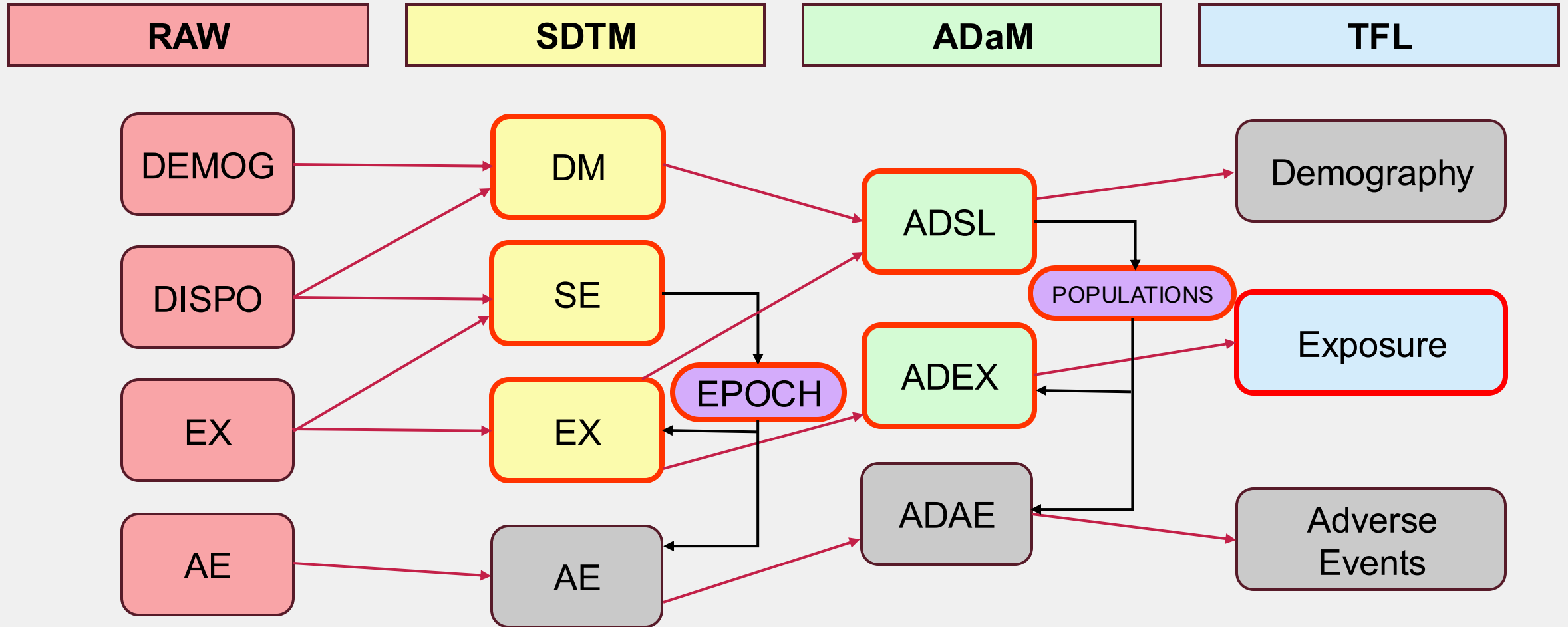
Case Study – Subset Rerun



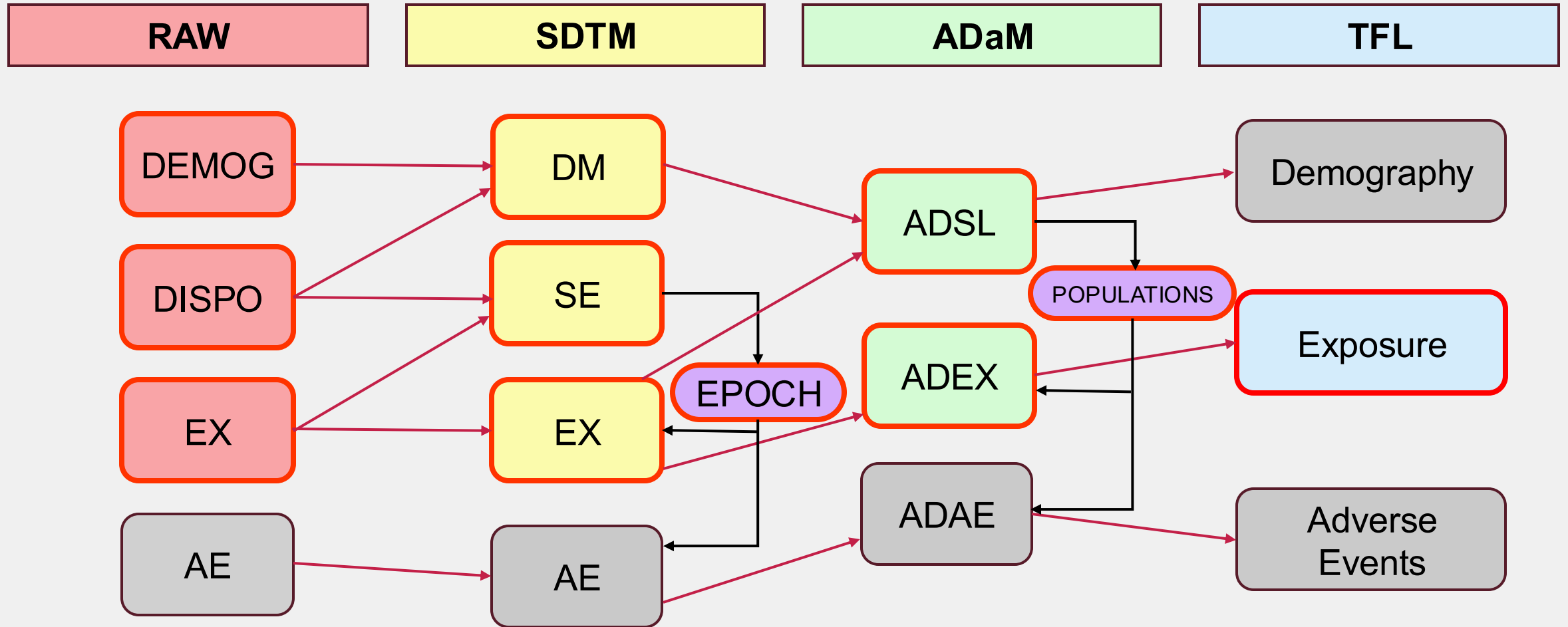
Case Study – Subset Rerun



Case Study – Subset Rerun



Case Study – Subset Rerun



Thank You

Any questions?

Contact:

rhys.mccrosson@phastar.com

simon.purkess@phastar.com

