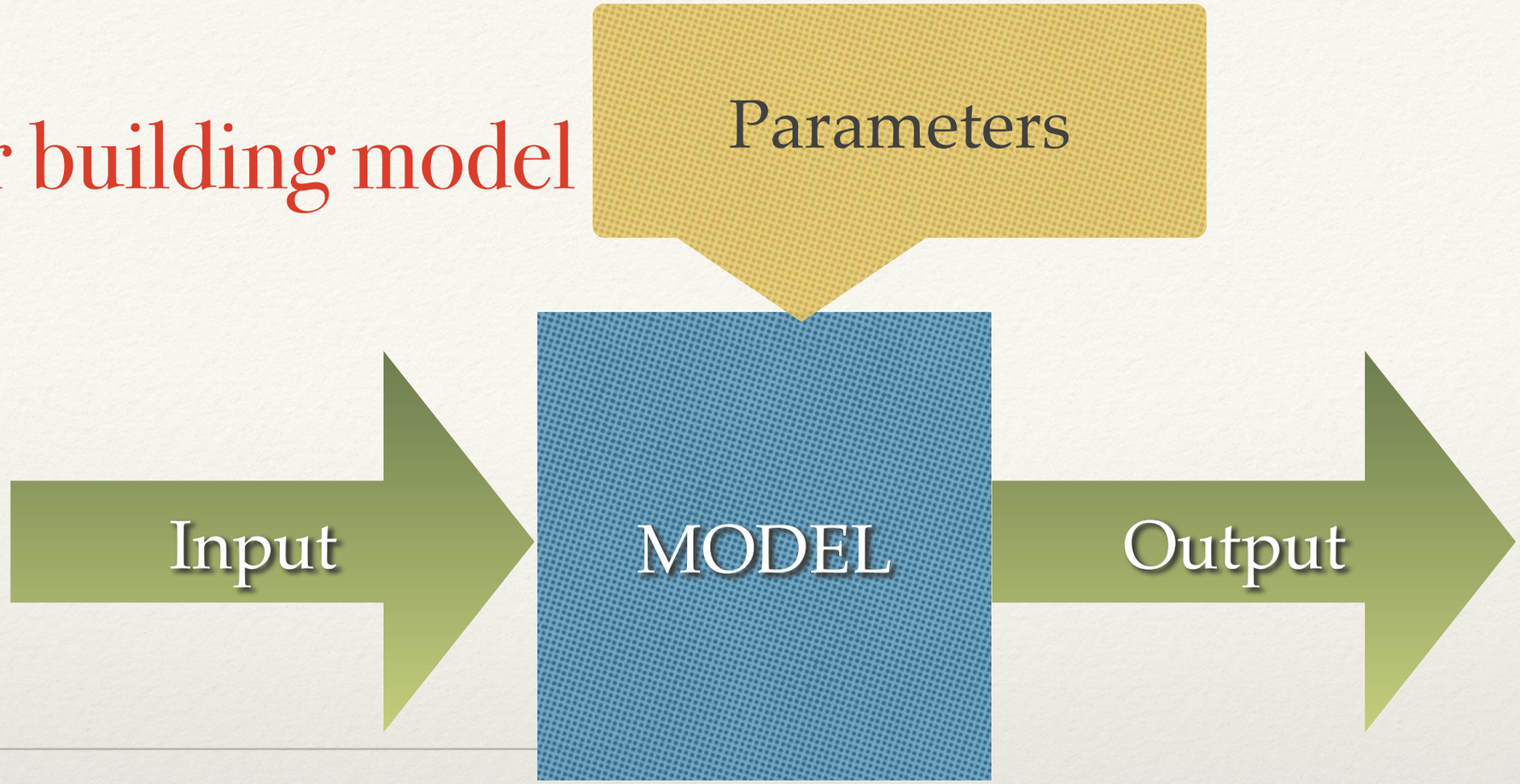
Steps for building model



1. Design conceptual model
2. Translate conceptual model into set of discrete tasks
3. Choose programming language
4. Define inputs (data type, units)
5. Define output (data type, units)
6. Define model structure
7. Write model
8. Document the model (meta data)
9. Test model

Best practices for model (software) development

- Common problems
  - Unreadable code (hard to understand, easy to forget how it works, hard to find errors, hard to expand)
  - Overly complex, disorganized code (hard to find errors; hard to modify-expand)
  - Insufficient testing (both during development and after)
  - Not tracking code changes (multiple versions, which is correct?)

Best practices for model (software) development

- Basic Solution
  - Structured practices that ensures
    - clear, readable code - with meaning variables names and comments to explain the workflow
    - modularity (organized “independent” building blocks)
    - testing as you go and after
    - code evolution is documented (version control)

Best practices for model (software) development

- Automated tools (useful for more complex code development
- (note that GP's often create complex models > 100 lines of code)
- Automated documentation
  - http://www.stack.nl/~dimitri/doxygen/
  - http://roxygen.org/roxygen2-manual.pdf
- Automated test case development
  - http://r-pkgs.had.co.nz/tests.html
- Automated code evolution tracking (Version Control)
  - https://github.com/

Discrete Tasks -

- Hierarchical in level of details
- Big chunks (coarse detail) -> progressively finer
- Note ‘tasks’ that need to be repeated
- All tasks should have inputs and outputs

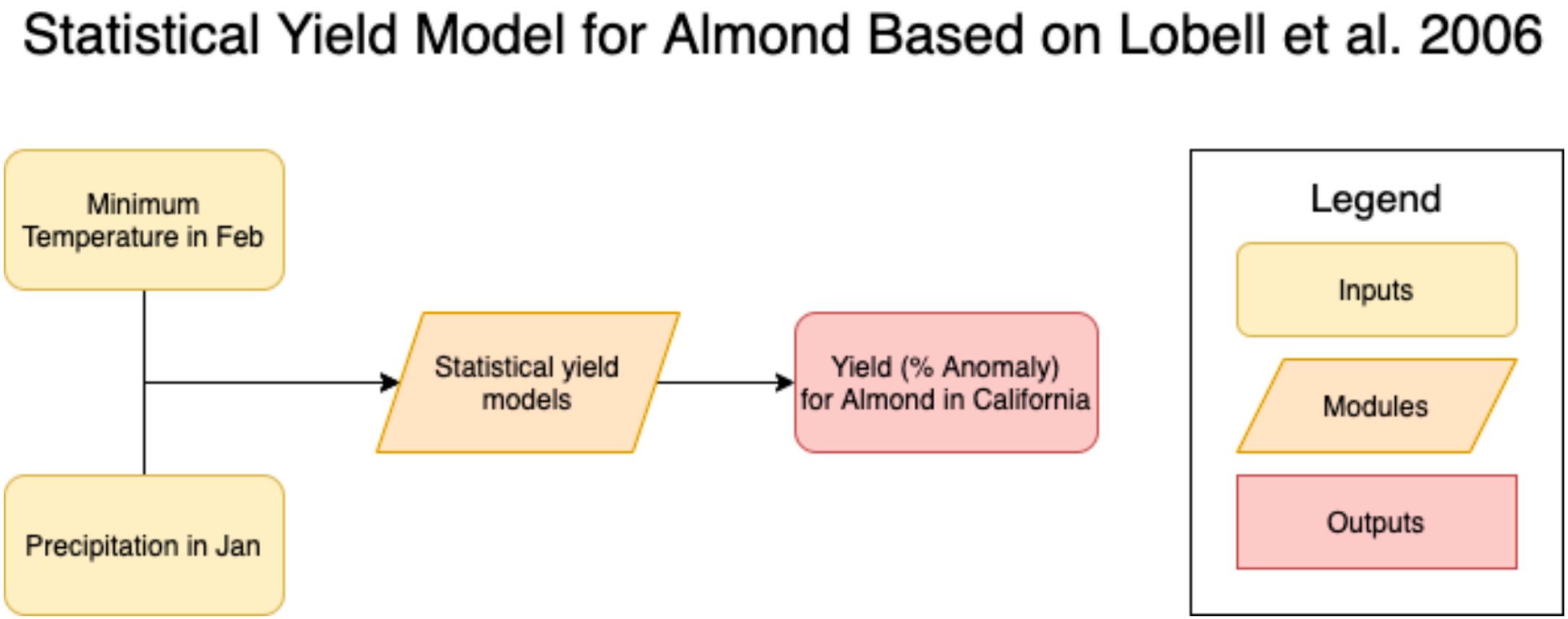
From conceptual model to flow chart/workflow

- Impact of smoke from fires on health of agricultural workers
- What is ‘smoke”
- What is “health”
- What is an “agricultural worker”. …leads to your conceptual model

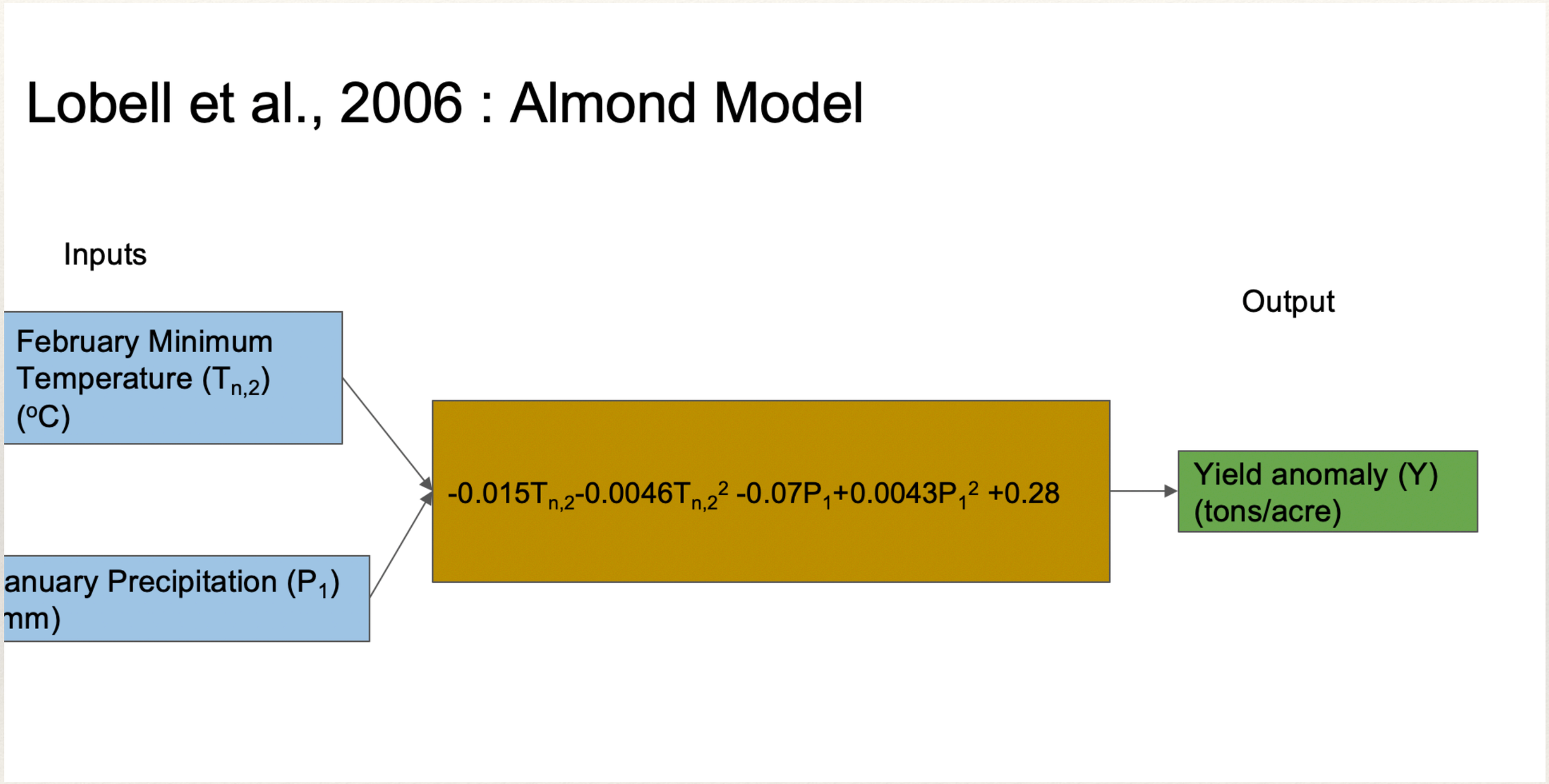
From conceptual model to flow chart/workflow

- Impact of climate on almond productivity
  - What is almond productivity?
  - What is climate?

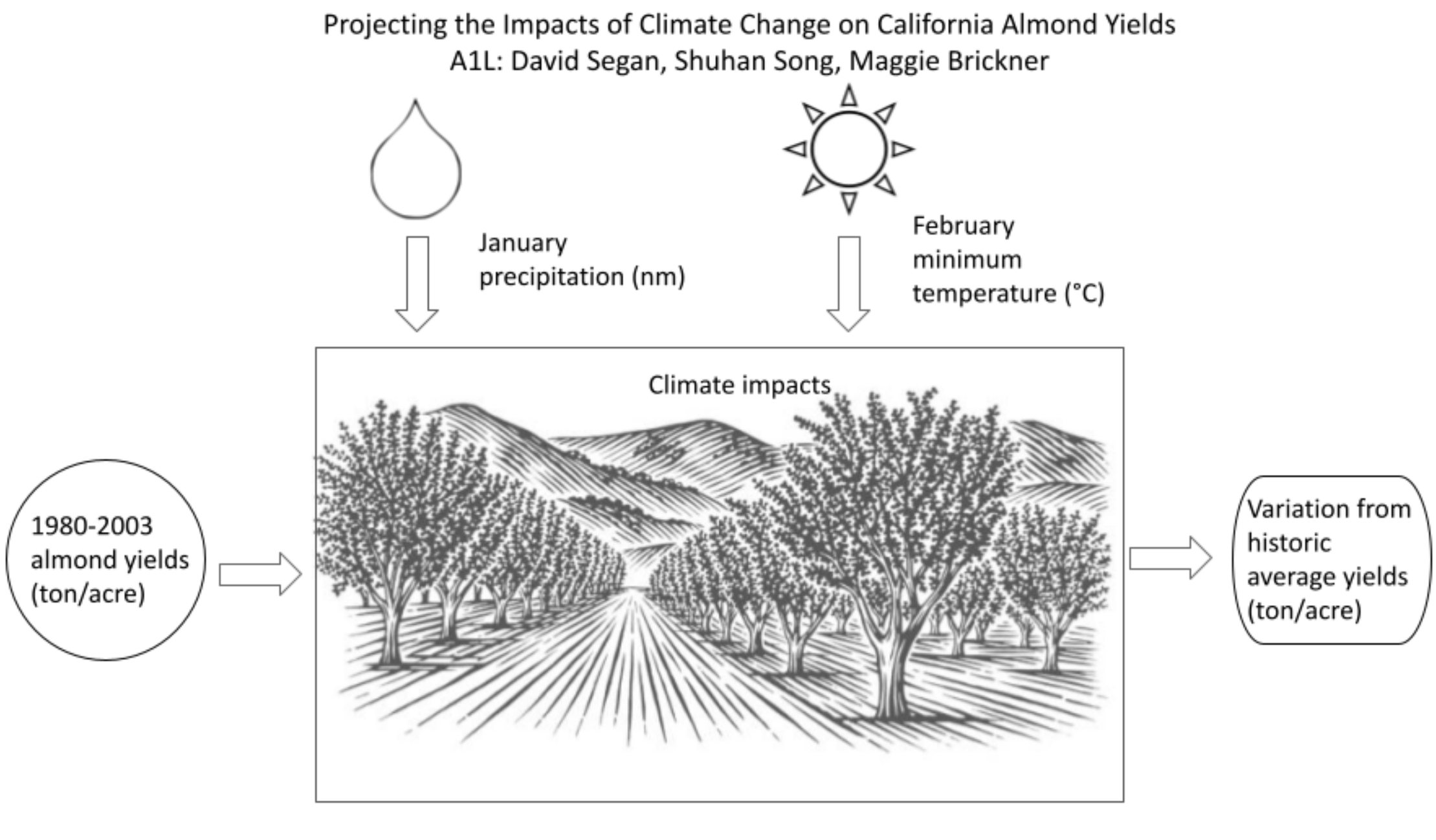
Simple, straightforward - but missing parameters



Simple, straightforward - but missing parameters

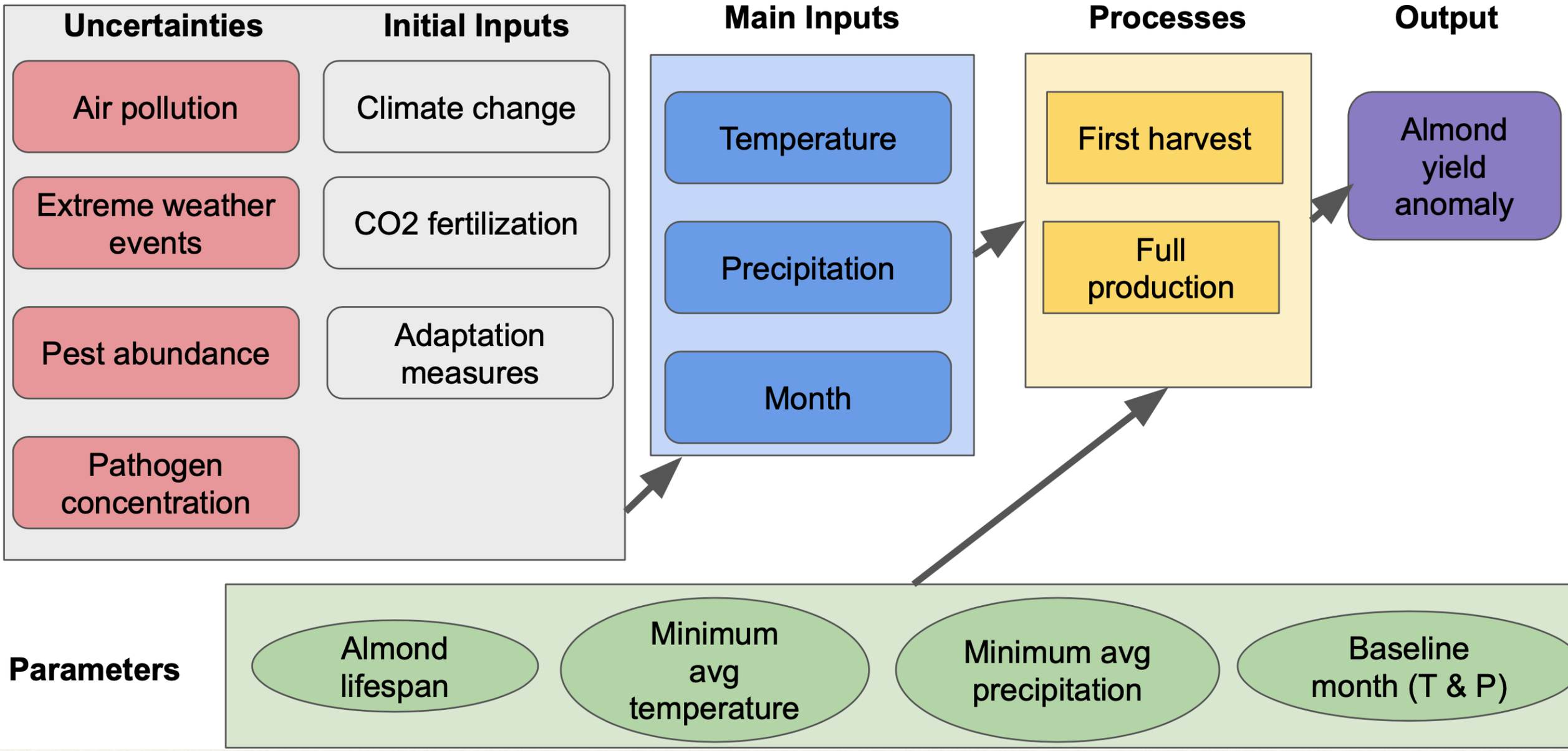


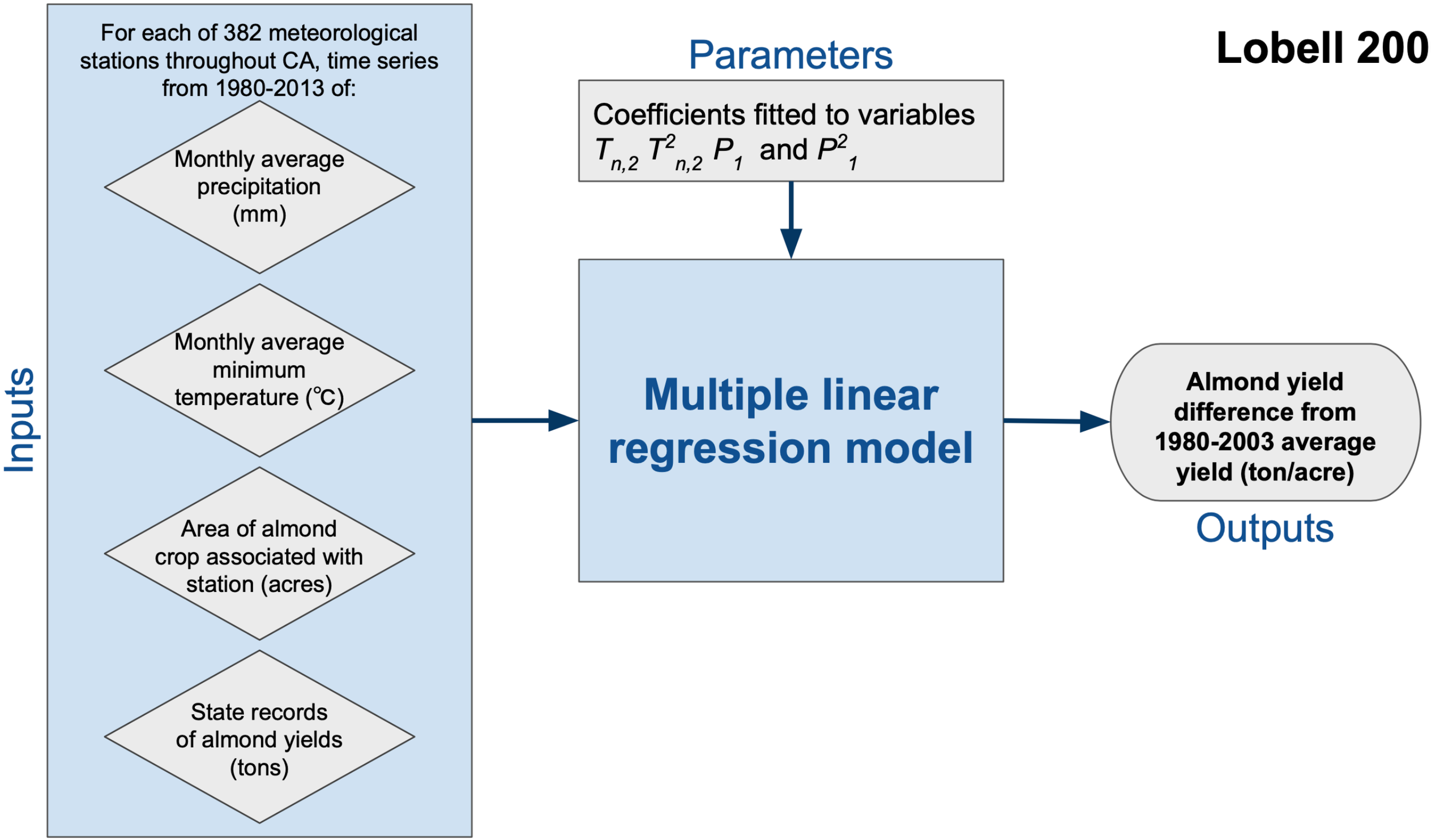
Because informative aesthetics is always a plus



$$Y = -0.015T_{n,2} - 0.0046T^2_{n,2} - 0.07P_1 + 0.0043P^2_1 + 0.28$$
$$R^2_{adj} = 0.88$$

Goal: Create a conceptual model of the almond yield anomaly using the equation in Lobell table 2.





Strategic use of parameters, climate extraction happens inside the model

Building Models

- Functions!
- The basic building blocks of models
- Functions can be written in all languages; in many languages (object-oriented) like C++, Python, functions are also objects
- Functions are the “boxes” of the model - the transfer function that takes inputs and returns outputs
- More complex models - made up of multiple functions; and nested functions (functions that call/user other functions)

Functions

- Write down:
  - all inputs and parameters
  - all outputs
- Decide what there data types, units, names should be
  - use descriptive names
  - use data types that will allow you to apply your function in many different cases

Functions

- Write down what the function will do - given different inputs and parameters
- simple
  - input (temperature); output (growth rate)
- more complex
  - inputs (temperature, organism type)
  - output (if animal, respiration; if plant, growth)

Implementing Functions in R

- Format for a basic function in R

#' documentation that describes inputs, outputs and what the function does

FUNCTION NAME = function(inputs, parameters) {

body of the function (manipulation of inputs)

return

}

In R, inputs and parameters are treated the same; but it is useful to think about them separately in designing the model - collectively they are sometimes referred to as arguments

ALWAYS USE Meaningful names for your function, its parameters and variables calculated within the function

Types of models: Example

- Input: Reservoir height and flow rate
- Output: Instantaneous power generation (W/s)
- Parameters: $K_{Efficiency}$, ϱ (density of water), g (acceleration due to gravity)

$P = \varrho * h * r * g * K_{Efficiency};$

P is Power in watts, ϱ is the density of water (~1000 kg/m3), h is height in meters, r is flow rate in cubic meters per second, g is acceleration due to gravity of 9.8 m/s2, $K_{Efficiency}$ is a coefficient of efficiency ranging from 0 to 1.

This is a static (one point in time), deterministic, lumped (one place) model; its more or less physically based

Building Models

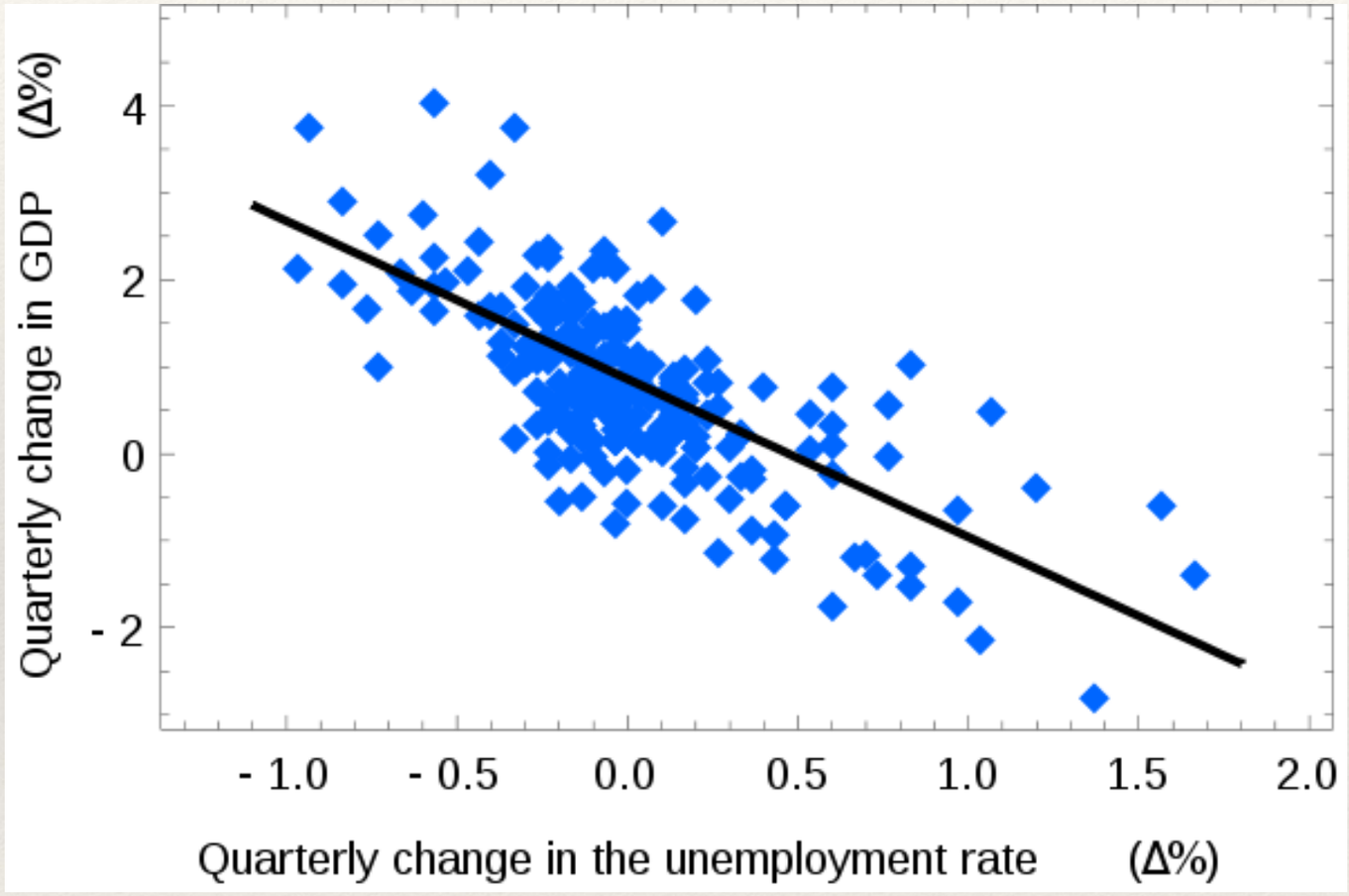
- Inputs/parameters are height, flow, rho, g, and K
- For some (particularly parameters) we provide default values by assigning them a value (e.g Keff = 0.8), but we can overwrite these
- Body is the equations between { and }
- *return* tells R what the output is

```
power_gen = function(height, flow, rho=1000, g=9.8, Keff=0.8) {

result = rho * height * flow * g * Keff
return(result)
}
```

Building Models

- Another example: Okuns Law (conceptual, abstract model) - try implementing this as a function



Graph of US quarterly data (not annualized) from 1947 through 2002 estimates a form of the difference version of Okun's law: %**Change GNP = .856 - 1.827*(Change Unemployment Rate).** R^2 of .504. Differences from other results are partly due to the use of quarterly data

http://en.wikipedia.org/wiki/Okun%27s_law *Write the function in R and add some error che*

```
#' Okuns Law
#'
#' function uses Okuns Law to estimate the quarterly change in GDP,
#’ from the quarterly change in unemployment
#' @param delta.unemploy Change in Unemployment Rate as percent
#' @param slope Slope of linear relationship Default is -1.827
#' @param intercept Intercep of linear relationship Default is 0.856
#' @examples
#' okun(delta.unemploy=3)
#' @references
#' Okun, Arthur, M, Potential GNP, its measurement and significance (1962).
Cowles Foundation, Yale University.
#' \url{http://cowles.econ.yale.edu/P/cp/p01b/p0190.pdf}


okun = function(deltaunemploy, intercep=0.856) {

deltaGDP = deltaunemploy*slope+intercep
return(deltaGDP)
}
```

What’s wrong with this model?

```
#' Okuns Law
#'
#' function uses Okuns Law to estimate the quarterly change in GDP,
#’ from the quarterly change in unemployment
#' @param deltaunemploy Change in Unemployment Rate as percent
#' @param slope Slope of linear relationship Default is -1.827
#' @param intercept Intercep of linear relationship Default is 0.856
#' @examples
#' okun(deltaunemploy=3)
#' @references
#' Okun, Arthur, M, Potential GNP, its measurement and significance (1962).
Cowles Foundation, Yale University.
#' \url{http://cowles.econ.yale.edu/P/cp/p01b/p0190.pdf}


okun = function(deltaunemploy, slope=-1.827, intercep=0.856) {

deltaGDP = deltaunemploy*slope+intercep

return(deltaGDP)
}
```

Usage

```
> okun(2)
[1] -2.798
> okun(-2)
[1] 4.51
> okun(-2, slope=-1.9)
[1] 4.656
```

Best practices for model (software) development

- Let us change our traditional attitude to the construction of programs: Instead of imagining that our main task is to instruct a computer what to do, let us concentrate rather on explaining to humans what we want the computer to do. -- Donald E. Knuth, Literate Programming, 1984

- Developing readable (by PEOPLE) code and documenting what you are doing is essential

- “When was the last time you spent a pleasant evening in a comfortable chair, reading a good program?”— Bentley (1986)

Documentation!

- Makes it easier for you to fix issues, refine and easier for others to use
- For functions (model) you should always include the following in your code
  - The goal of the model
  - Model inputs (description, name and units)
  - Model output (description, name and units)
  - Where the transfer function comes from (source papers)
  - In line explanation for key steps
- Always use meaningful names for functions, variables, inputs and outputs

Building Models:Documentation

- There is also a standard format for documentation that can be read by automatic programs (roxygen2) - an R package that generate "standard" R documentation - manual or help pages
- These automated approaches for building documentation (like roxygen2) and meta data (descriptions of data sets) are increasingly common - so you should get into the practice of being structured in your approach to documentation
- We will use the conventions that work with roxygen2 - and then use this program to generate formal R documentation. Roxygen is similar to Doxygen which is used for C code…so its a widely used format