Integrated Assessment Modelling

Volker Krey

14 October 2024

NTNU course: Integrated Assessment Modelling (EP8900)

Overview

- Introduction to Linear Programming
- Good Practice Modeling (version control)
- Introduction to GAMS

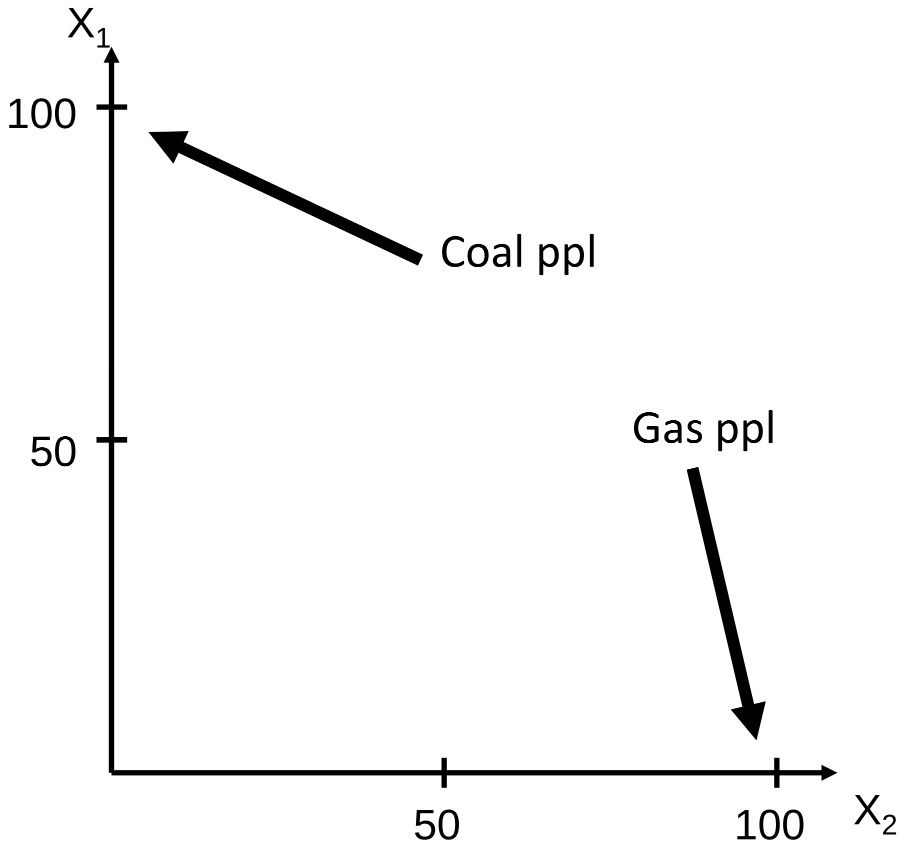
A short introduction to Linear Programming

Material based on lecture by Nebojša Nakićenović

A Simple Linear Program

Minimize costs of two power plants:

x_1 costs €20/MWh while x_2 costs €22/MWh

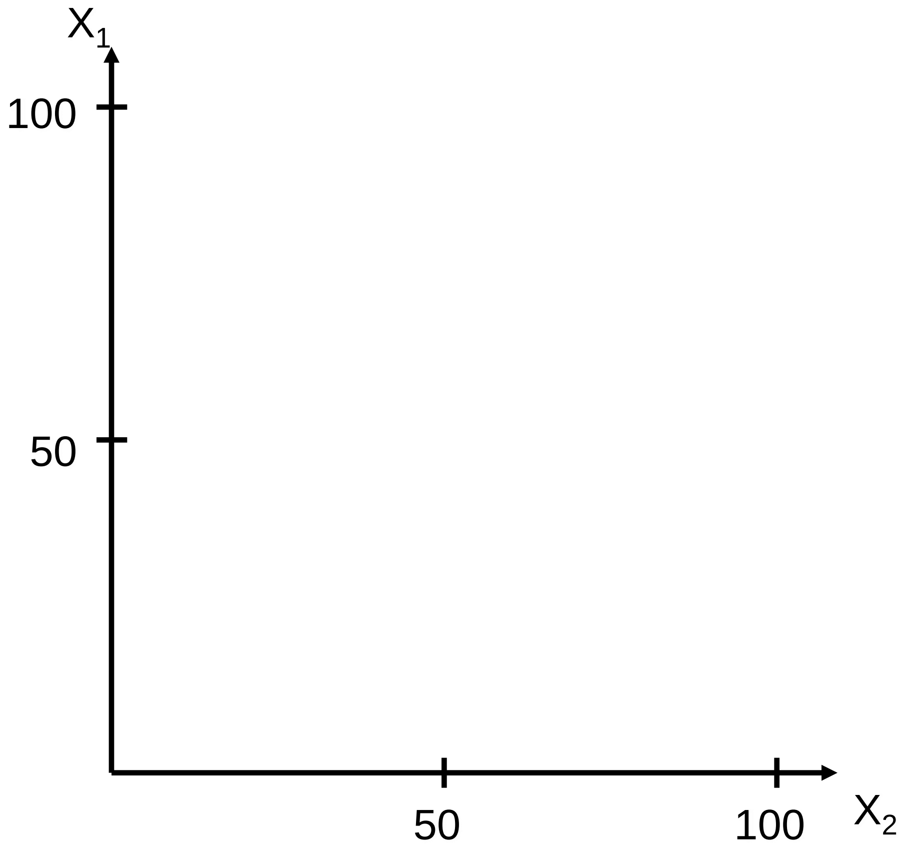


$x_1 \geq 0; x_2 \geq 0$

A Simple Linear Program

Minimize costs of two power plants:

x_1 costs €20/MWh while x_2 costs €22/MWh



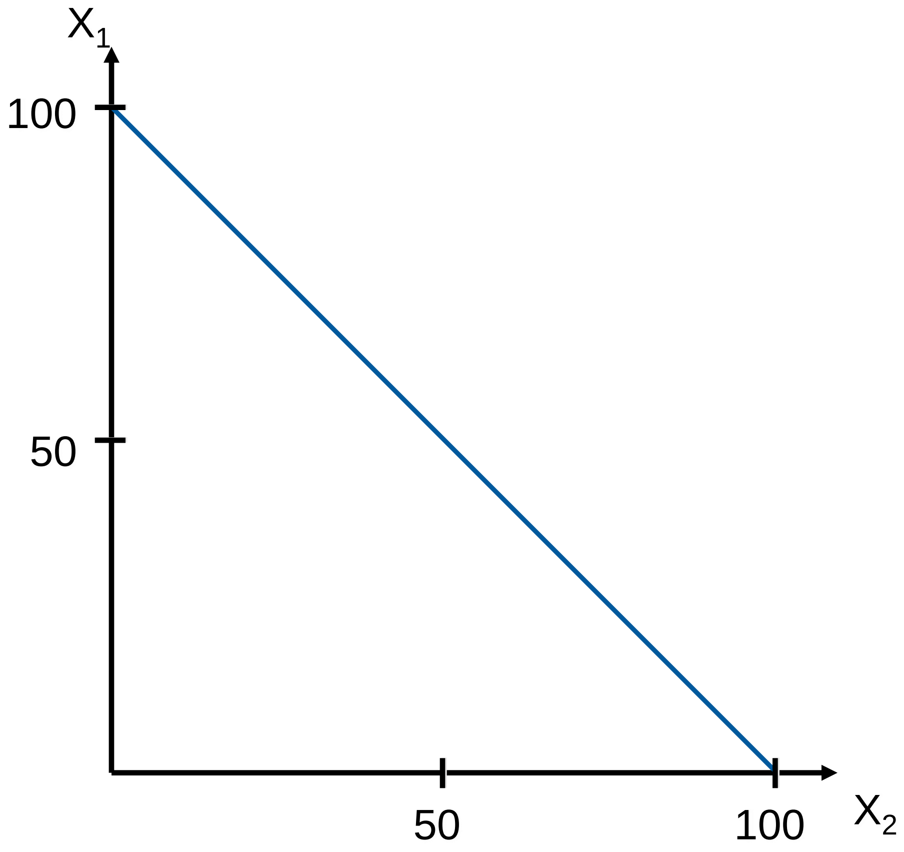
$x_1 + x_2 \geq 100$ (MWh)

$x_1 \geq 0; x_2 \geq 0$

A Simple Linear Program

Minimize costs of two power plants:

x_1 costs €20/MWh while x_2 costs €22/MWh



Time unit: 1h

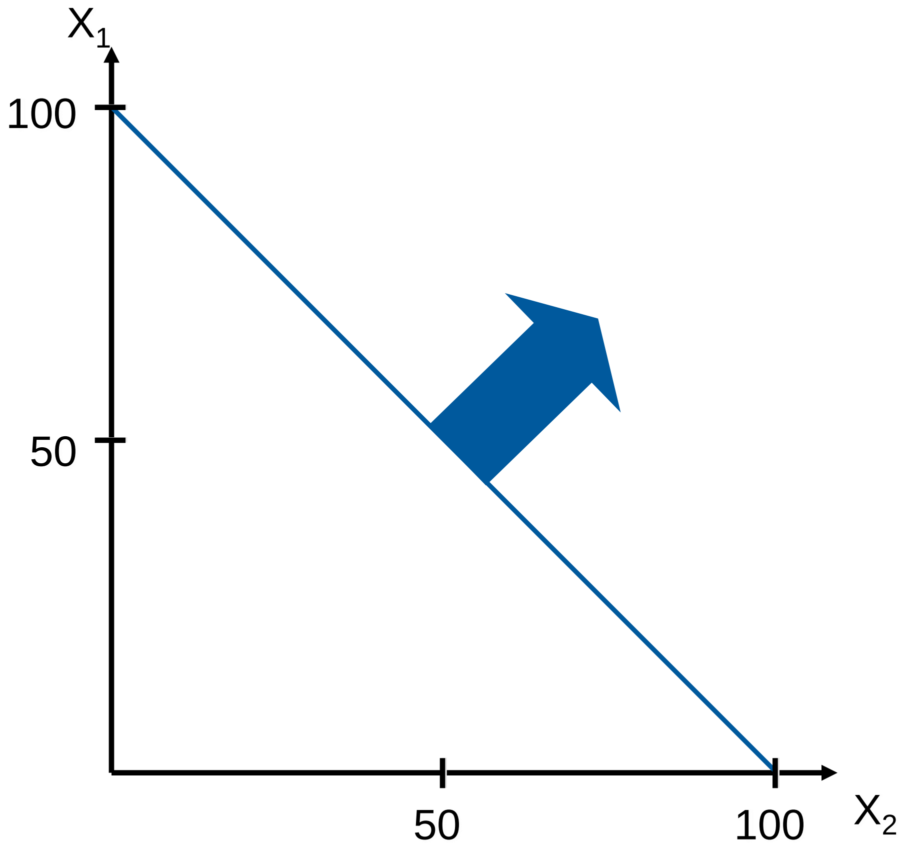
$x_1 + x_2 \geq 100$ (MWh)

$x_1 \geq 0$; $x_2 \geq 0$

A Simple Linear Program

Minimize costs of two power plants:

x_1 costs €20/MWh while x_2 costs €22/MWh



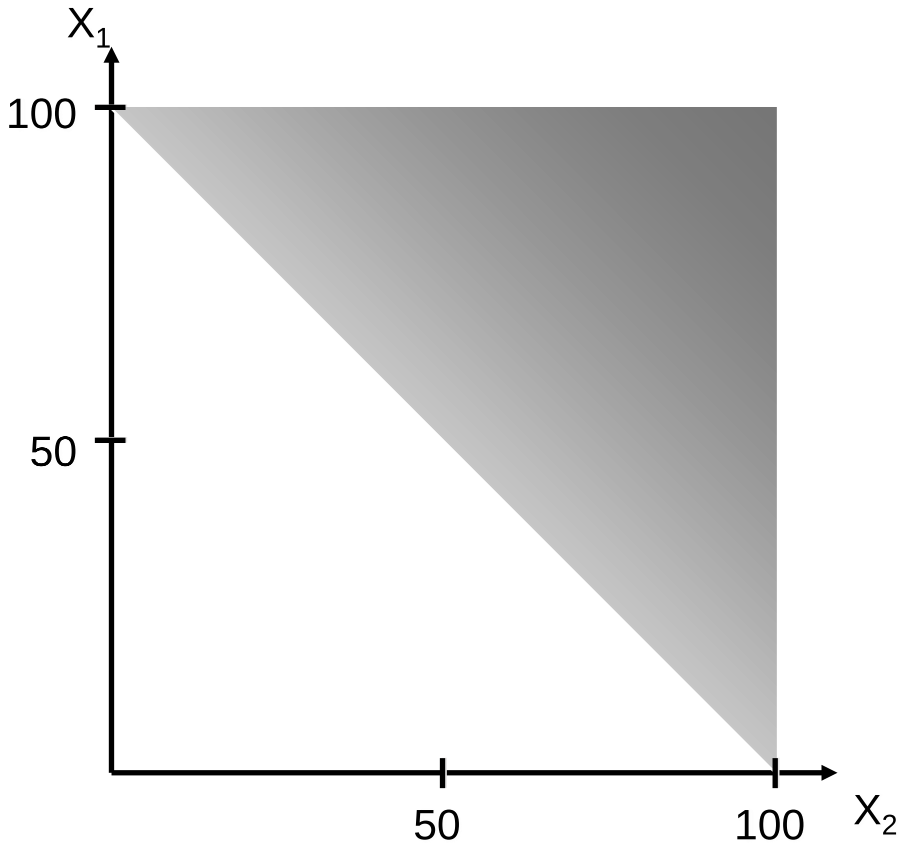
$x_1 + x_2 \geq 100$ (MWh)

$x_1 \geq 0; x_2 \geq 0$

A Simple Linear Program

Minimize costs of two power plants:

x_1 costs €20/MWh while x_2 costs €22/MWh



$x_1 + x_2 \geq 100$ (MWh)

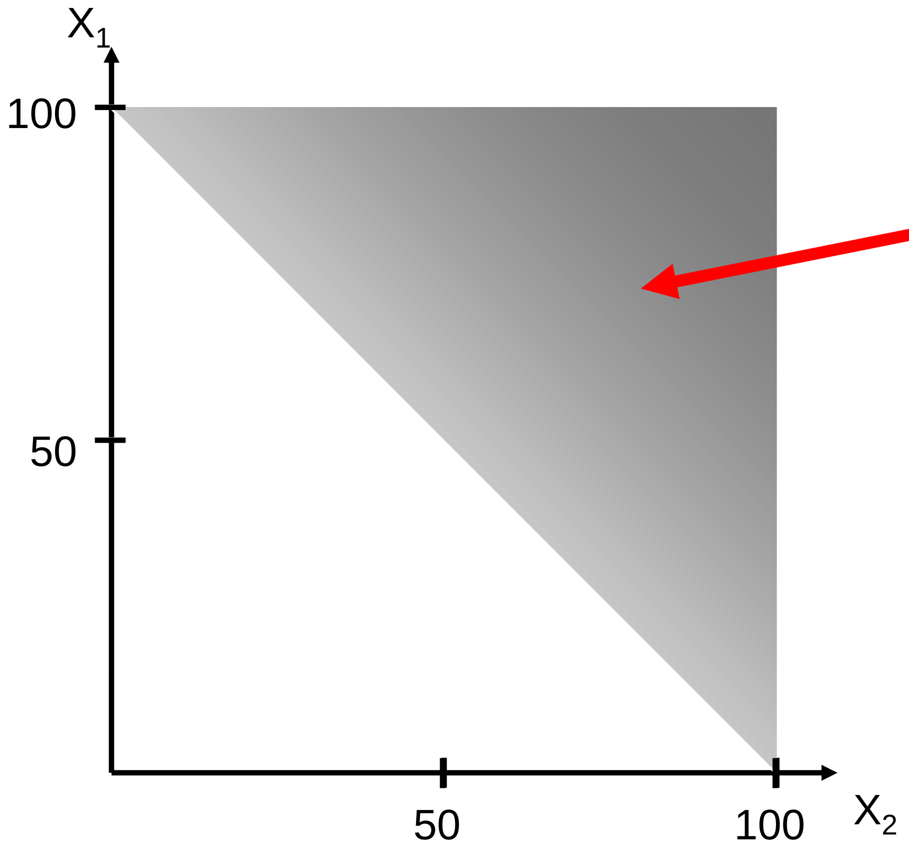
($x_1 \leq 100$; $x_2 \leq 100$)

$x_1 \geq 0$; $x_2 \geq 0$

A Simple Linear Program

Minimize costs of two power plants:

x_1 costs €20/MWh while x_2 costs €22/MWh



Feasible Region

$x_1 + x_2 \geq 100$ (MWh)

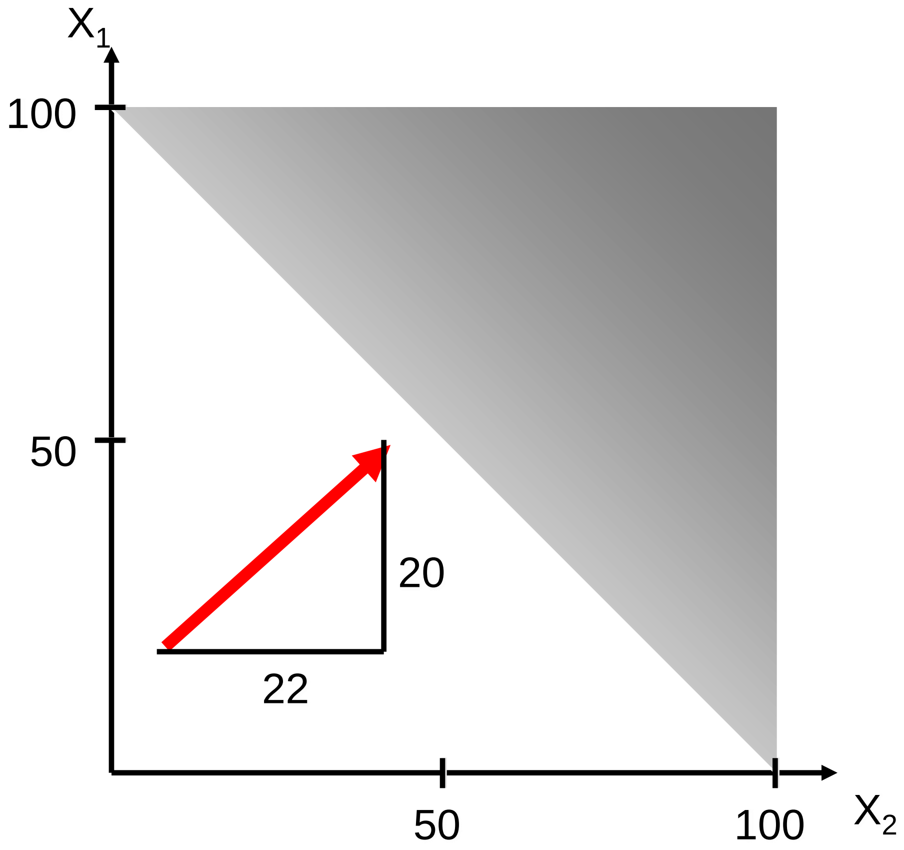
($x_1 \leq 100$; $x_2 \leq 100$)

$x_1 \geq 0$; $x_2 \geq 0$

A Simple Linear Program

Minimize costs of two power plants:

x_1 costs €20/MWh while x_2 costs €22/MWh



Min: $20x_1 + 22x_2$

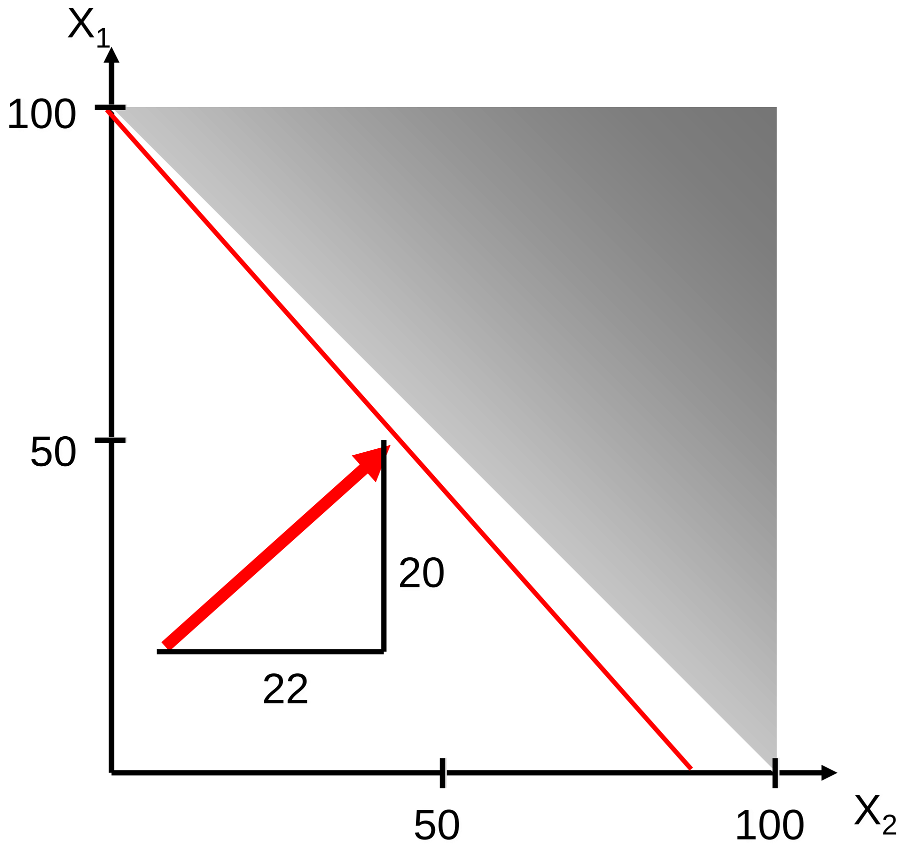
$x_1 + x_2 \geq 100$ (MWh)

$x_1 \geq 0; x_2 \geq 0$

A Simple Linear Program

Minimize costs of two power plants:

x_1 costs €20/MWh while x_2 costs €22/MWh



Min: $20x_1 + 22x_2$

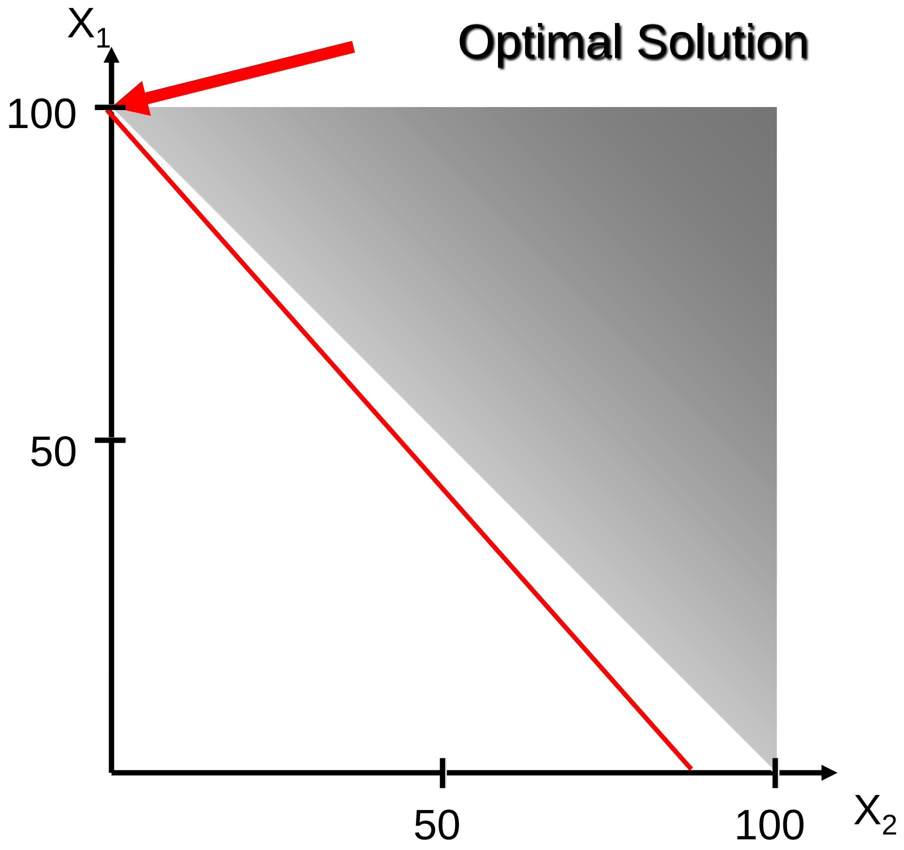
$x_1 + x_2 \geq 100$ (MWh)

$x_1 \geq 0; x_2 \geq 0$

A Simple Linear Program

Minimize costs of two power plants:

x_1 costs €20/MWh while x_2 costs €22/MWh



Min: $20x_1 + 22x_2$

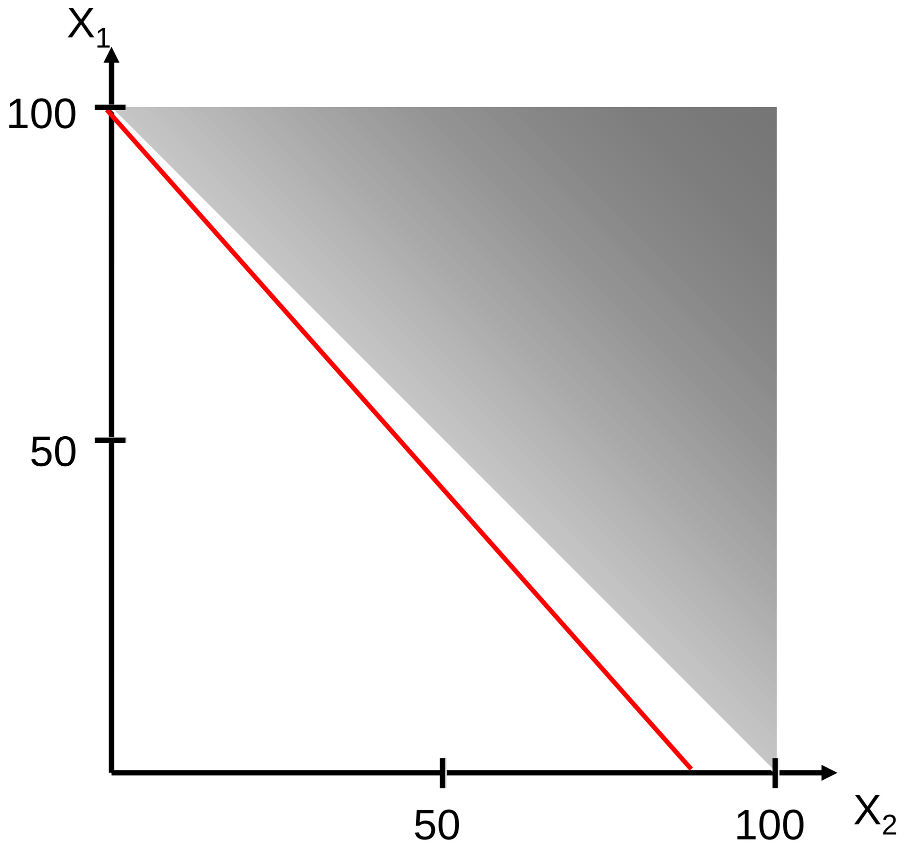
$x_1 + x_2 \geq 100$ (MWh)

$x_1 \geq 0; x_2 \geq 0$

A Simple Linear Program

Minimize costs of two power plants:

x_1 costs €20/MWh while x_2 costs €22/MWh



Min: $20x_1 + 22x_2$

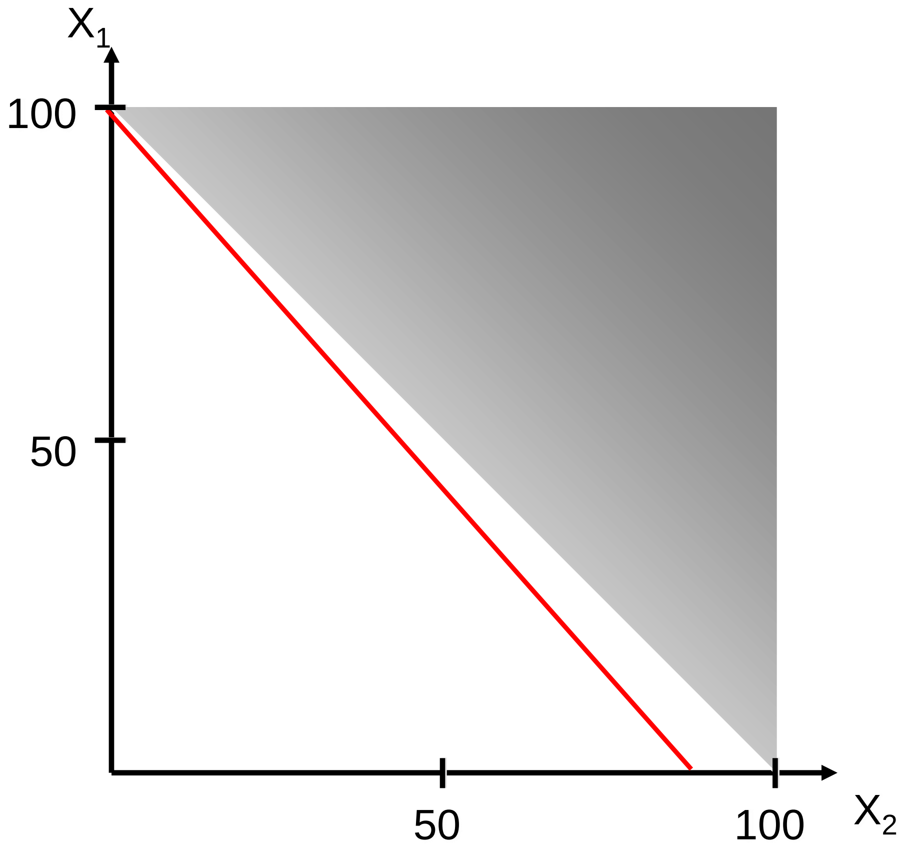
$x_1 + x_2 \geq 100$ (MWh)

$x_1 \geq 0$; $x_2 \geq 0$

A Simple Linear Program

Minimize costs of two power plants:

x_1 costs €20/MWh while x_2 costs €22/MWh



Min: $20x_1 + 22x_2$

$x_1 + x_2 \geq 100$ (MWh)

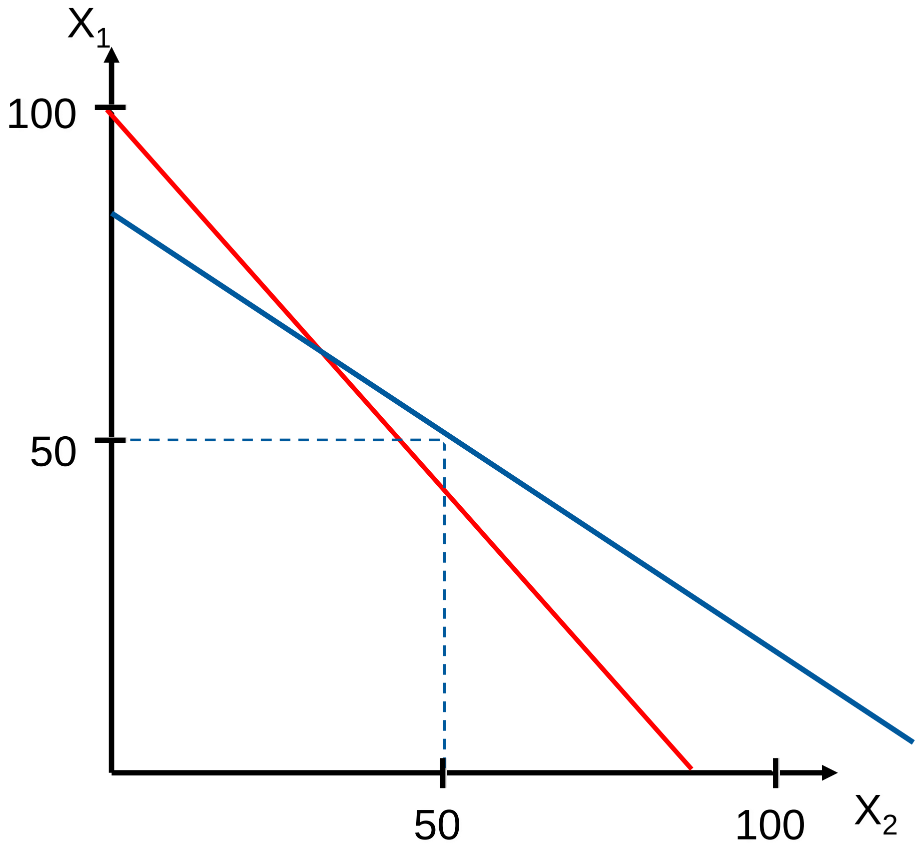
$1.5x_1 + x_2 \leq 125$ (tCO2)

$x_1 \geq 0; x_2 \geq 0$

A Simple Linear Program

Minimize costs of two power plants:

x_1 costs €20/MWh while x_2 costs €22/MWh



Min: $20x_1 + 22x_2$

$x_1 + x_2 \geq 100$ (MWh)

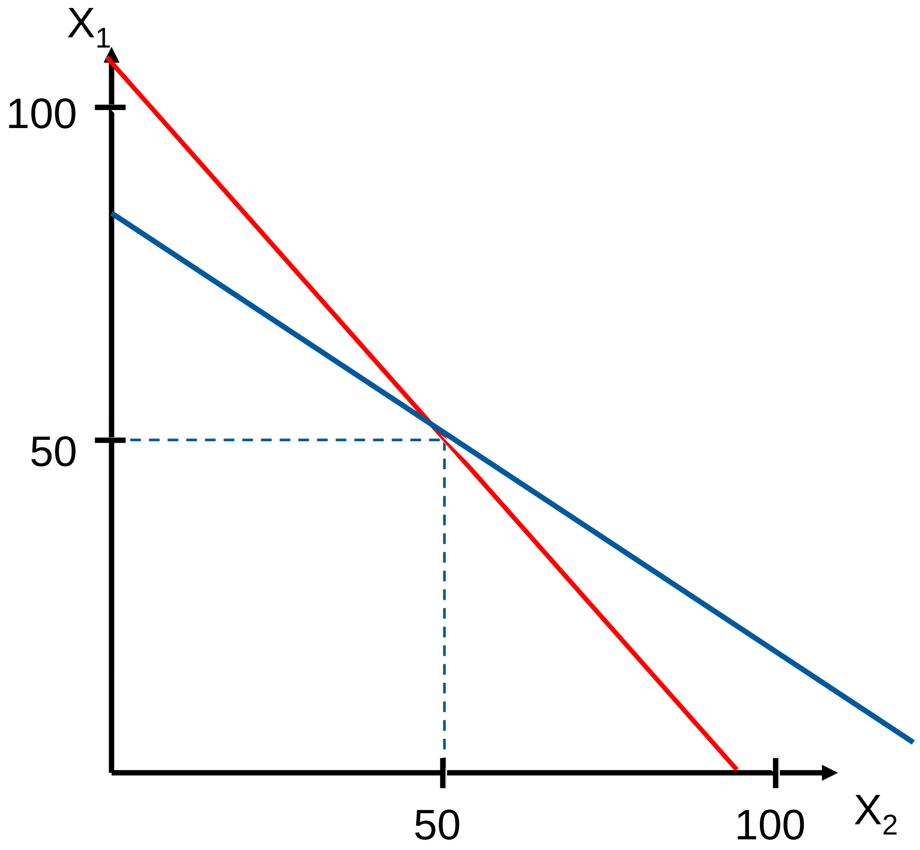
$1.5x_1 + x_2 \leq 125$ (tCO2)

$x_1 \geq 0; x_2 \geq 0$

A Simple Linear Program

Minimize costs of two power plants:

x_1 costs €20/MWh while x_2 costs €22/MWh



Min: $20x_1 + 22x_2$

$x_1 + x_2 \geq 100$ (MWh)

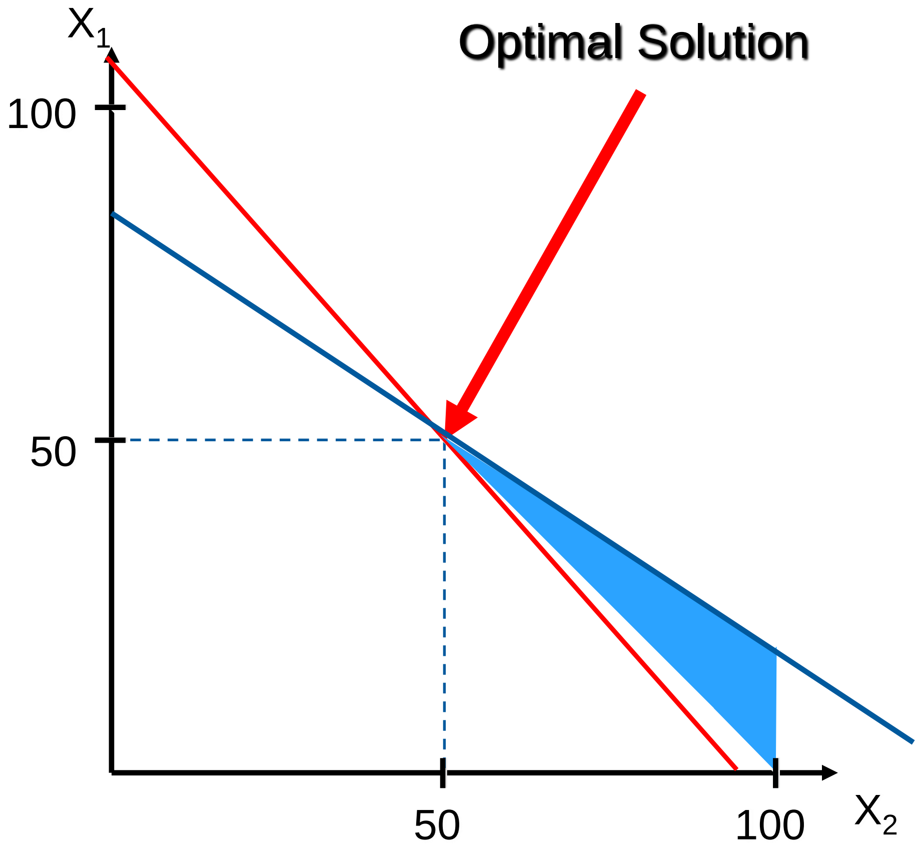
$1.5x_1 + x_2 \leq 125$ (tCO2)

$x_1 \geq 0; x_2 \geq 0$

A Simple Linear Program

Minimize costs of two power plants:

x_1 costs €20/MWh while x_2 costs €22/MWh



Min: $20x_1 + 22x_2$

$x_1 + x_2 \geq 100$ (MWh)

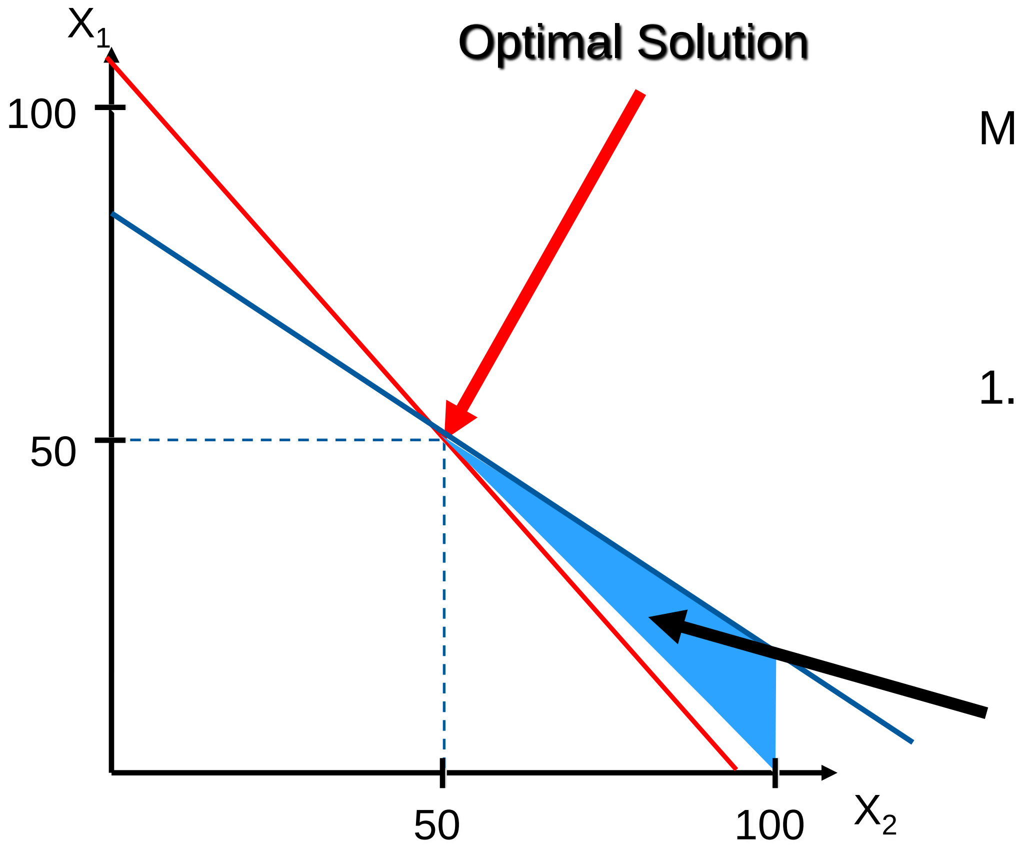
$1.5x_1 + x_2 \leq 125$ (tCO2)

$x_1 \geq 0; x_2 \geq 0$

A Simple Linear Program

Minimize costs of two power plants:

x_1 costs €20/MWh while x_2 costs €22/MWh



Optimal Solution

Overproduction

Min: $20x_1 + 22x_2$

$x_1 + x_2 \geq 100$ (MWh)

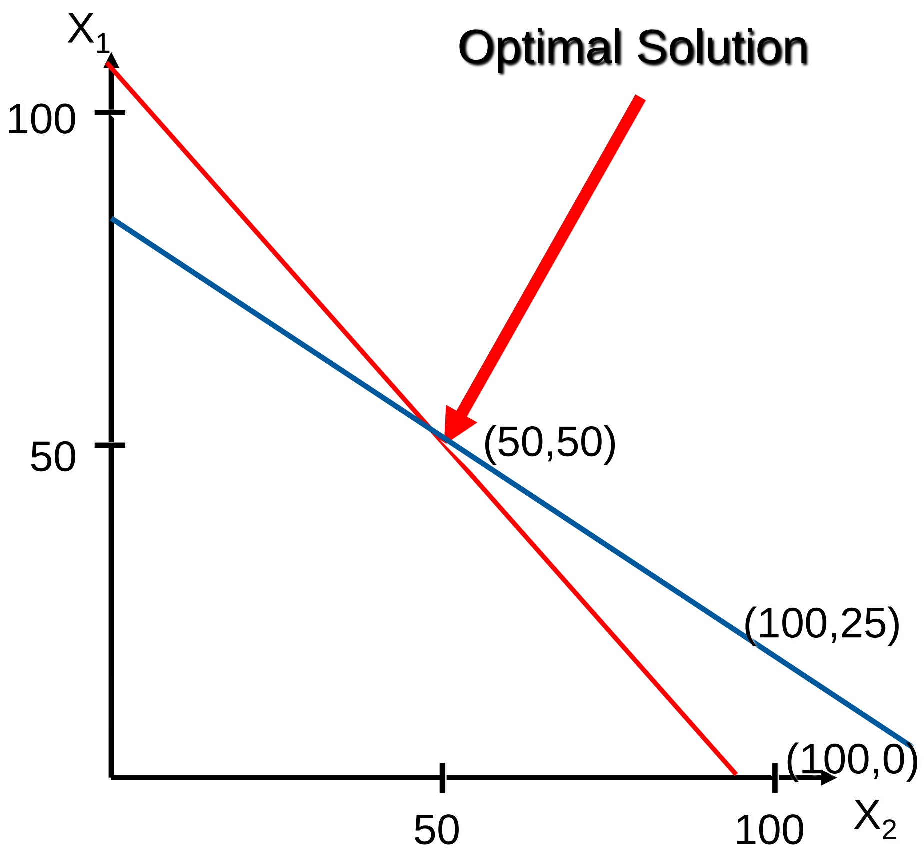
$1.5x_1 + x_2 \leq 125$ (tCO2)

$x_1 \geq 0; x_2 \geq 0$

A Simple Linear Program

Minimize costs of two power plants:

x_1 costs €20/MWh while x_2 costs €22/MWh



Min: $20x_1 + 22x_2$

$x_1 + x_2 \geq 100$ (MWh)

$1.5x_1 + x_2 \leq 125$ (tCO2)

$x_1 \geq 0; x_2 \geq 0$

Key Properties of Linear Programs

- The optimum point is always at a feasible corner point
- If a corner point feasible solution has an objective functions value that is better than or equal to all adjacent corner points feasible solutions, than it is optimal
- There are a finite number of corner point feasible solutions

The Standard Linear Program

- The objective function must be maximized
- All constraints are less or equal (≤) type
- All constraint right hand sides are nonnegative
- All variables are restricted to non-negativity

Objective function:

$$\text{Max. } G = p_1x_1 + p_2x_2 + \ldots.. + p_nx_n$$

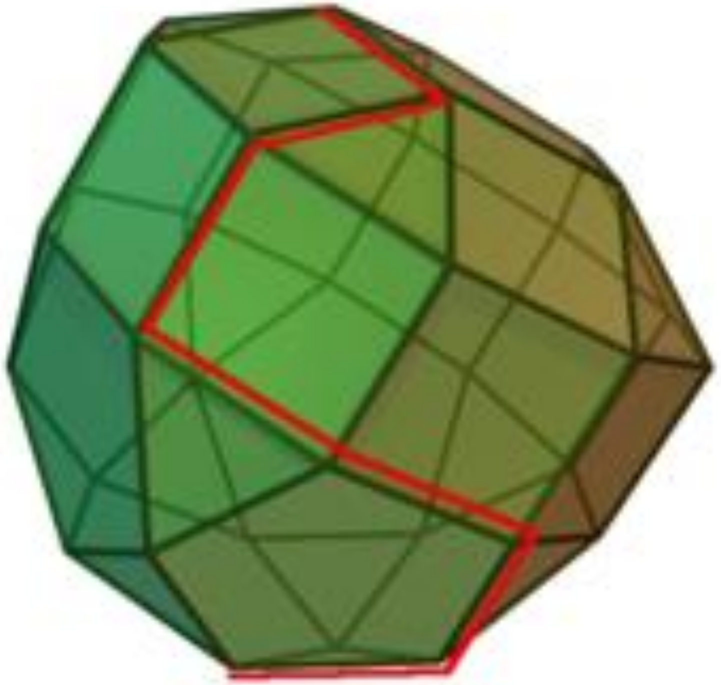
NB:

$$\begin{aligned}
a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n &\le b_1 \\
a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n &\le b_2 \\
\vdots \qquad\qquad\qquad & \quad \vdots \\
a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n &\le b_m
\end{aligned}$$

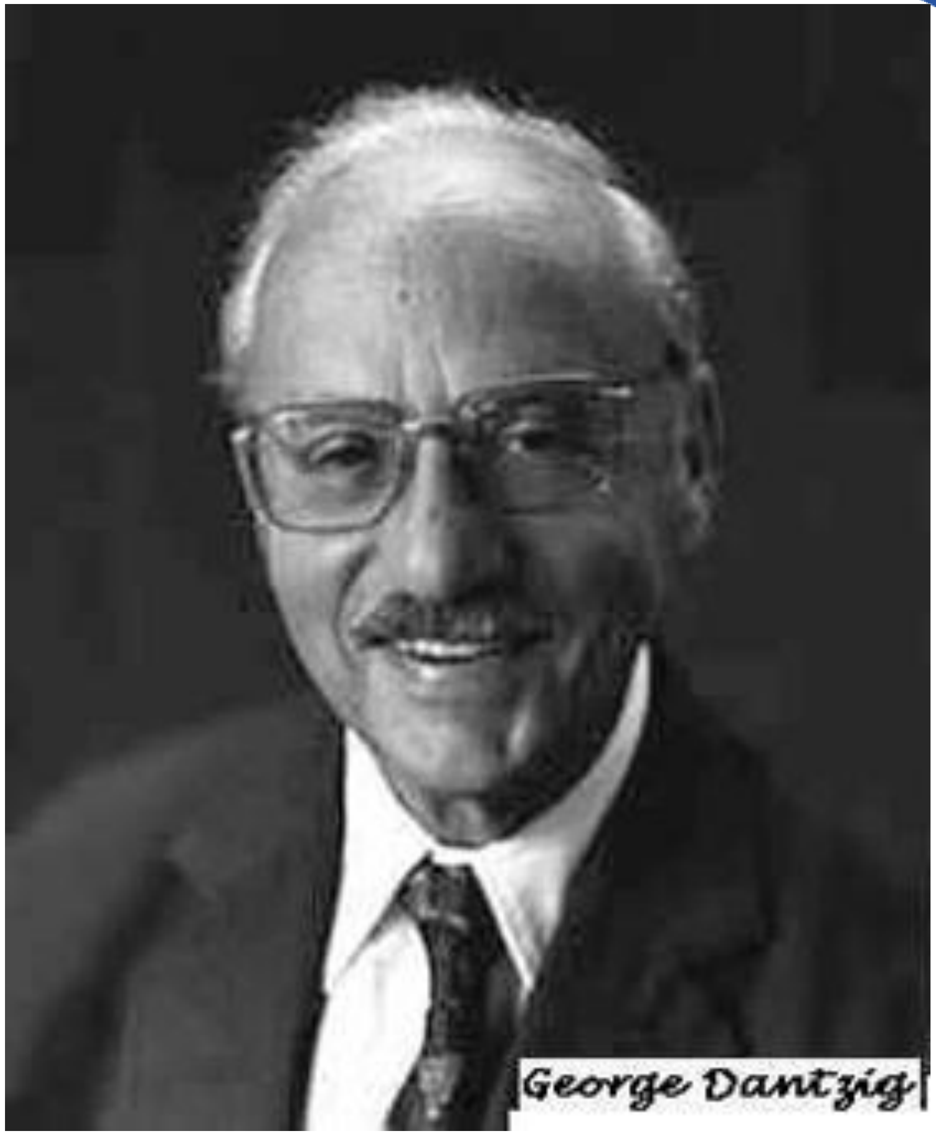
Simplex Algorithm

- Developed by George Dantzig (1914-2005)
- Published in 1947

Simplex Method in 3 dimensions:



Source: Wikipedia, 2007



Simplex Algorithm

- Find any corner point feasible solution, e.g., in a standard LP origin is such a solution
- Repeatedly move to a better adjacent corner point feasible solution until no further better adjacent corner point feasible solutions can be found

Good scientific programming practice: tools for reproducible science

Material courtesy of Paul Kishimoto, Daniel Huppmann, Matthew Gidden

Some Basic Questions

- Is it still science if it's not reproducible?
- Who should be able to reproduce it?
  - ⇒ Another scientist?
  - ⇒ A reviewer?
  - ⇒ Your colleague?
  - ⇒ You?
- How long should something be reproducible?
  - ⇒ Can you reproduce a figure from a paper you wrote a year ago?
  - ⇒ Can you reproduce a figure for a paper in review?
- How long does it take to reproduce?
- What does it mean if you don't get the same answer?

Best Practices

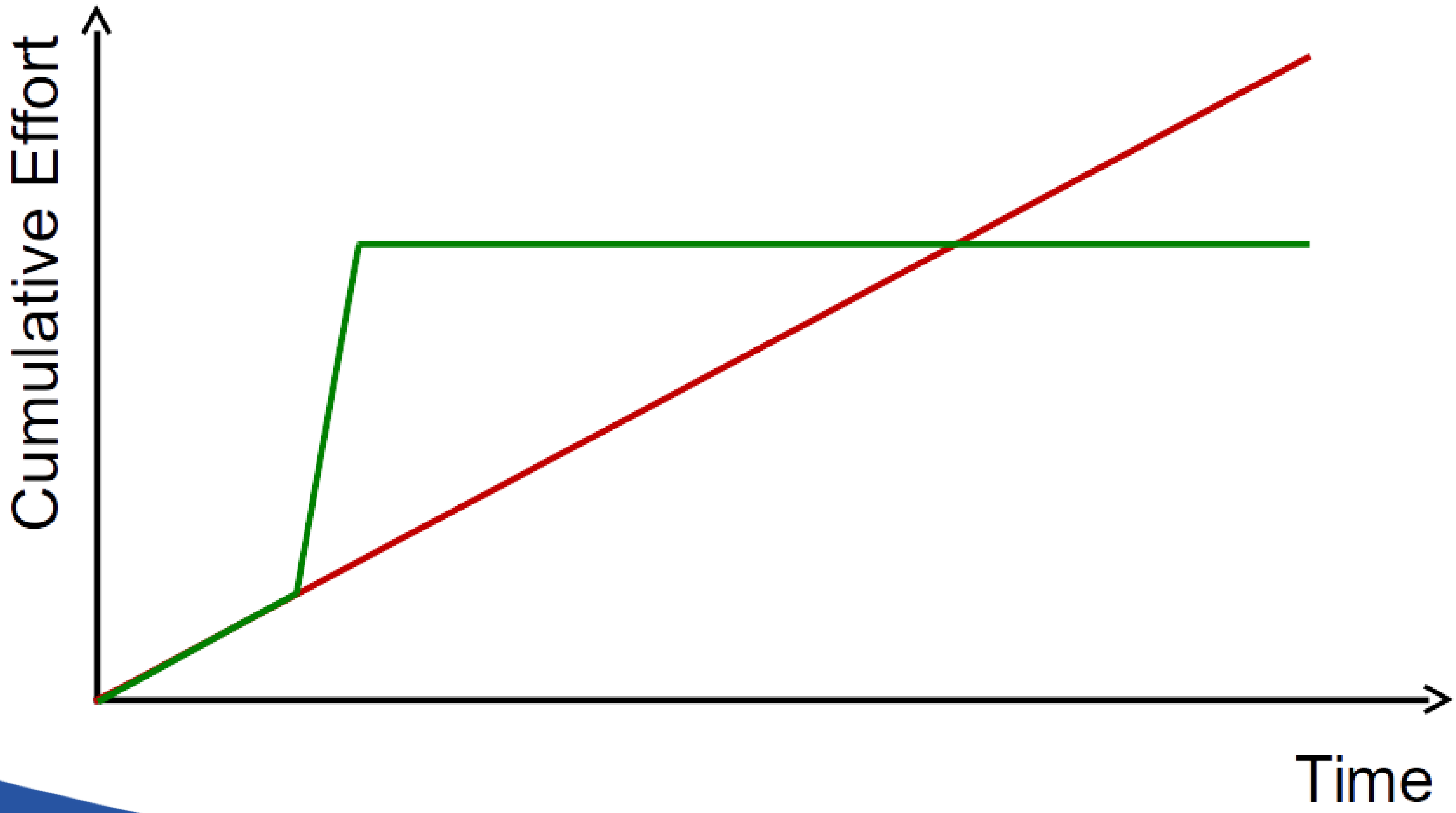
- When writing code
  - ⇒ Write software for people, not computers
  - ⇒ Don’t repeat yourself
  - ⇒ Make it correct, then make it fast
  - ⇒ Make incremental changes
  - ⇒ Use consistent style

- To be reproducible
  - ⇒ Automate repetitive tasks
  - ⇒ Use version control
  - ⇒ Plan for mistakes
- Working as a team
  - ⇒ Document design, purpose, and assumptions
  - ⇒ Conduct code reviews
  - ⇒ Use available release & management tools

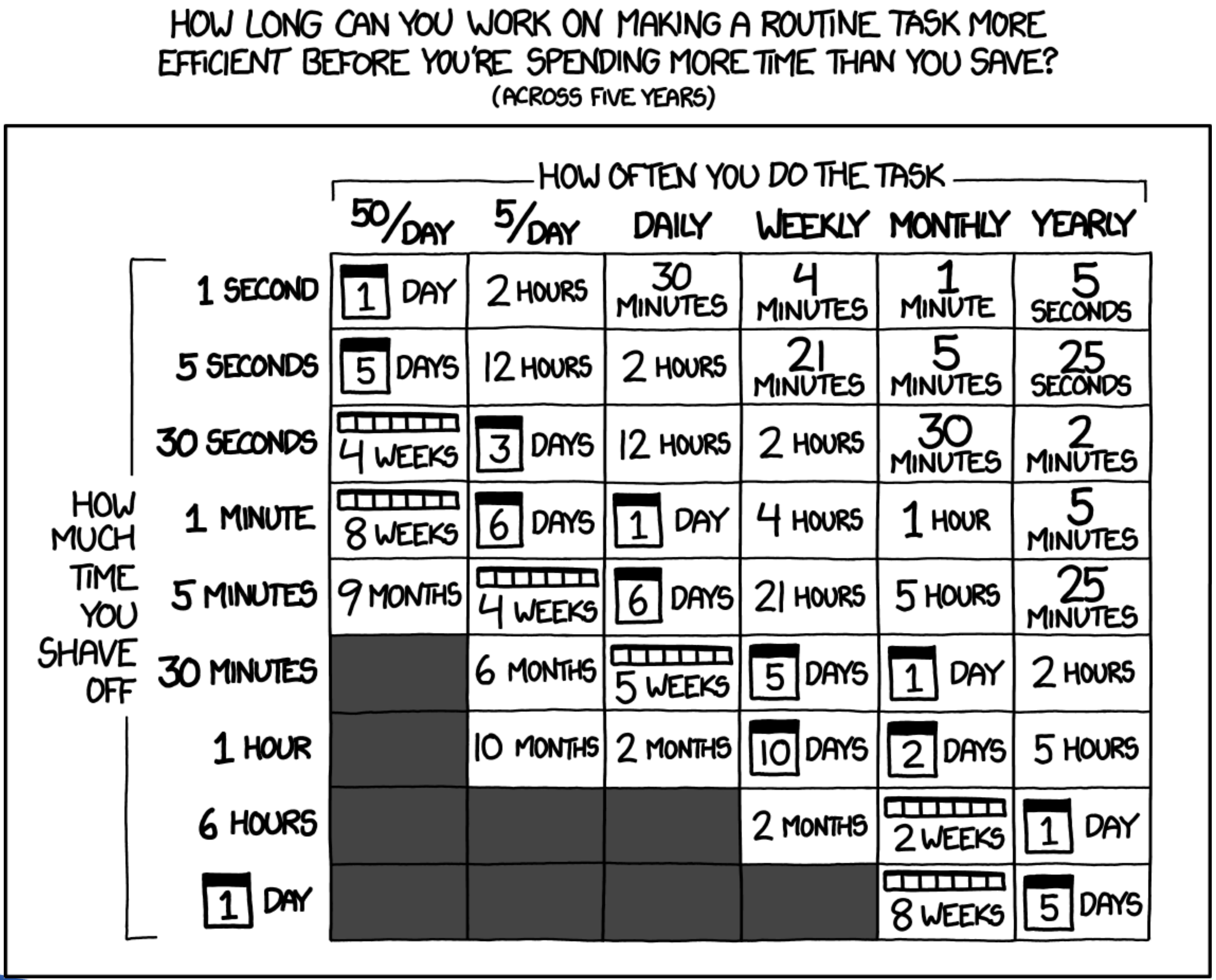
Why Follow the Personal Best Practices?

Your closest collaborator is you six months ago, but she/he doesn’t reply to emails.

Productivity vs. Best Practices Overhead



Time allocation for increasing efficiency through automation



HOW LONG CAN YOU WORK ON MAKING A ROUTINE TASK MORE EFFICIENT BEFORE YOU'RE SPENDING MORE TIME THAN YOU SAVE?
(ACROSS FIVE YEARS)

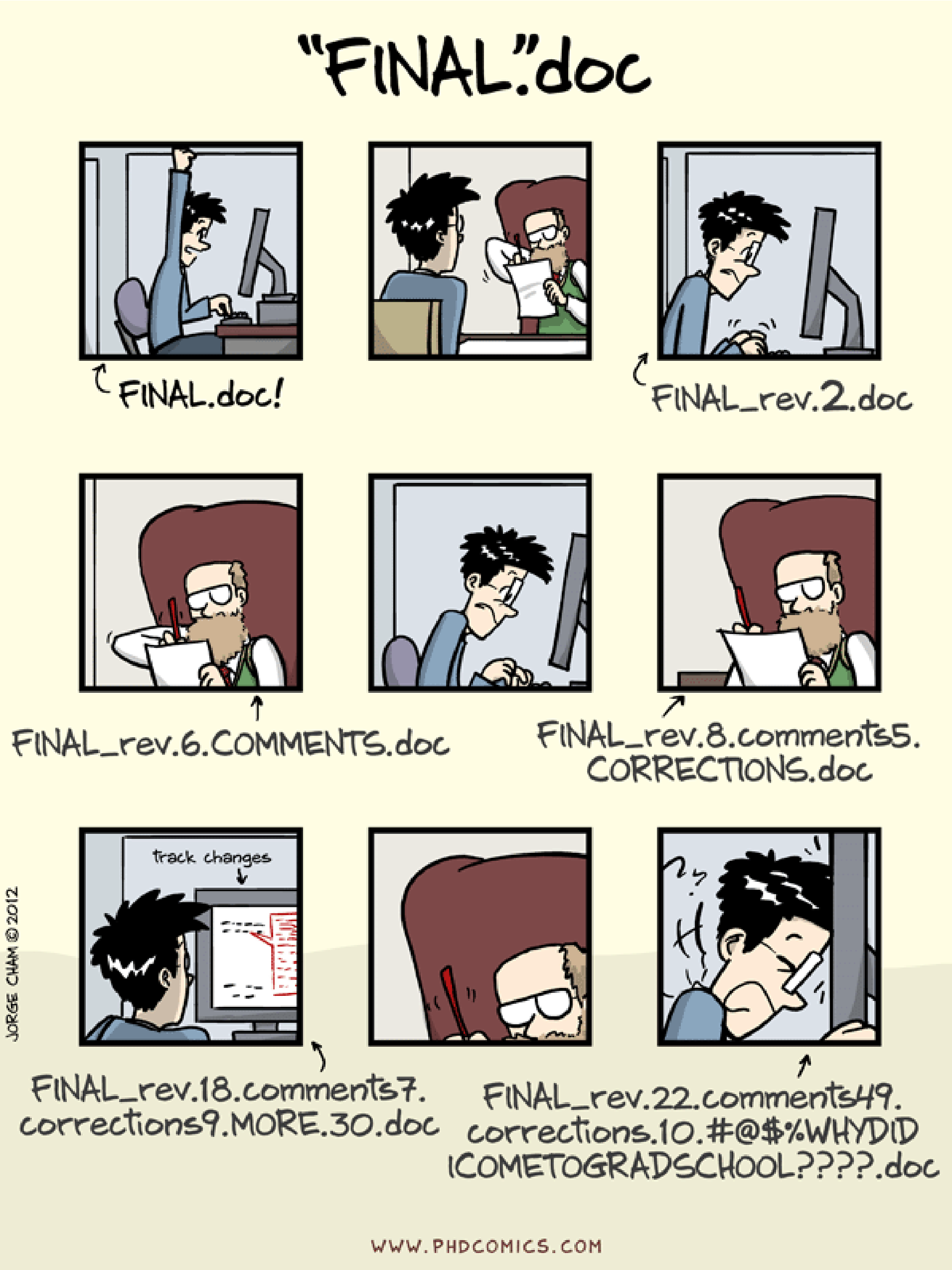
HOW MUCH TIME YOU SHAVE OFF	HOW OFTEN YOU DO THE TASK					
	50/DAY	5/DAY	DAILY	WEEKLY	MONTHLY	YEARLY
1 SECOND	1 DAY	2 HOURS	30 MINUTES	4 MINUTES	1 MINUTE	5 SECONDS
5 SECONDS	5 DAYS	12 HOURS	2 HOURS	21 MINUTES	5 MINUTES	25 SECONDS
30 SECONDS	4 WEEKS	3 DAYS	12 HOURS	2 HOURS	30 MINUTES	2 MINUTES
1 MINUTE	8 WEEKS	6 DAYS	1 DAY	4 HOURS	1 HOUR	5 MINUTES
5 MINUTES	9 MONTHS	4 WEEKS	6 DAYS	21 HOURS	5 HOURS	25 MINUTES
30 MINUTES		6 MONTHS	5 WEEKS	5 DAYS	1 DAY	2 HOURS
1 HOUR		10 MONTHS	2 MONTHS	10 DAYS	2 DAYS	5 HOURS
6 HOURS				2 MONTHS	2 WEEKS	1 DAY
1 DAY					8 WEEKS	5 DAYS

xkcd by Randall Munroe

Version control using git & GitHub

Based on a lectures by Paul Kishimoto and Matthew Gidden

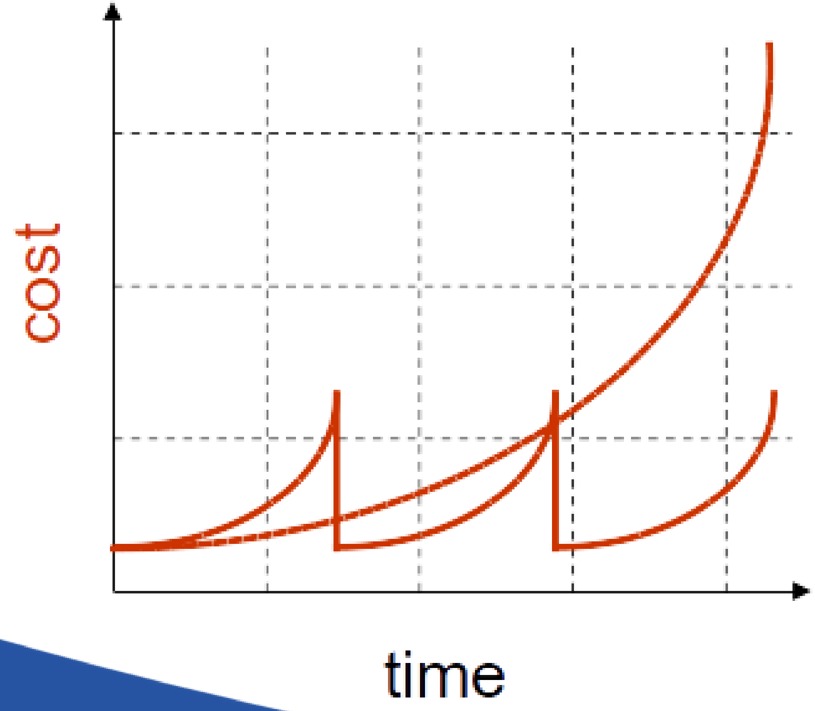
Version Control



Source: http://phdcomics.com/comics/archive.php?comicid=1531

Version Control

- Save your work in “units”
- Lower cognitive load
- Easier to find bugs



Two primary choices:

1. Centralized
   ⇒ SVN, etc.
2. Decentralized
   ⇒ git, etc.

Here we opt for *decentralized* because of its flexibility and existing tool base.

Version control systems

- Version control is the management of changes to documents, computer programs and other collections of information.
  - ⇒ Changes or states usually identified by a number or letter code.
  - ⇒ Each revision associated with a timestamp and author.
  - ⇒ Revisions can be compared, restored, and combined.
- Version control systems (VCS, "revision control systems", other names) are software that tracks and provide control over revisions.
  - ⇒ Automate repetitive, boring processes.
  - ⇒ These could be (often are!) done manually.
  - ⇒ But, because they are monotonous, mistakes are likely.
- Manage the chronological and sequential relationship between revisions.

`git`: a VCS

Several different VCS available. Some tools (e.g. Dropbox; MS Office "track changes") provides a subset of VCS-like features…but not suitable for models and scientific code.

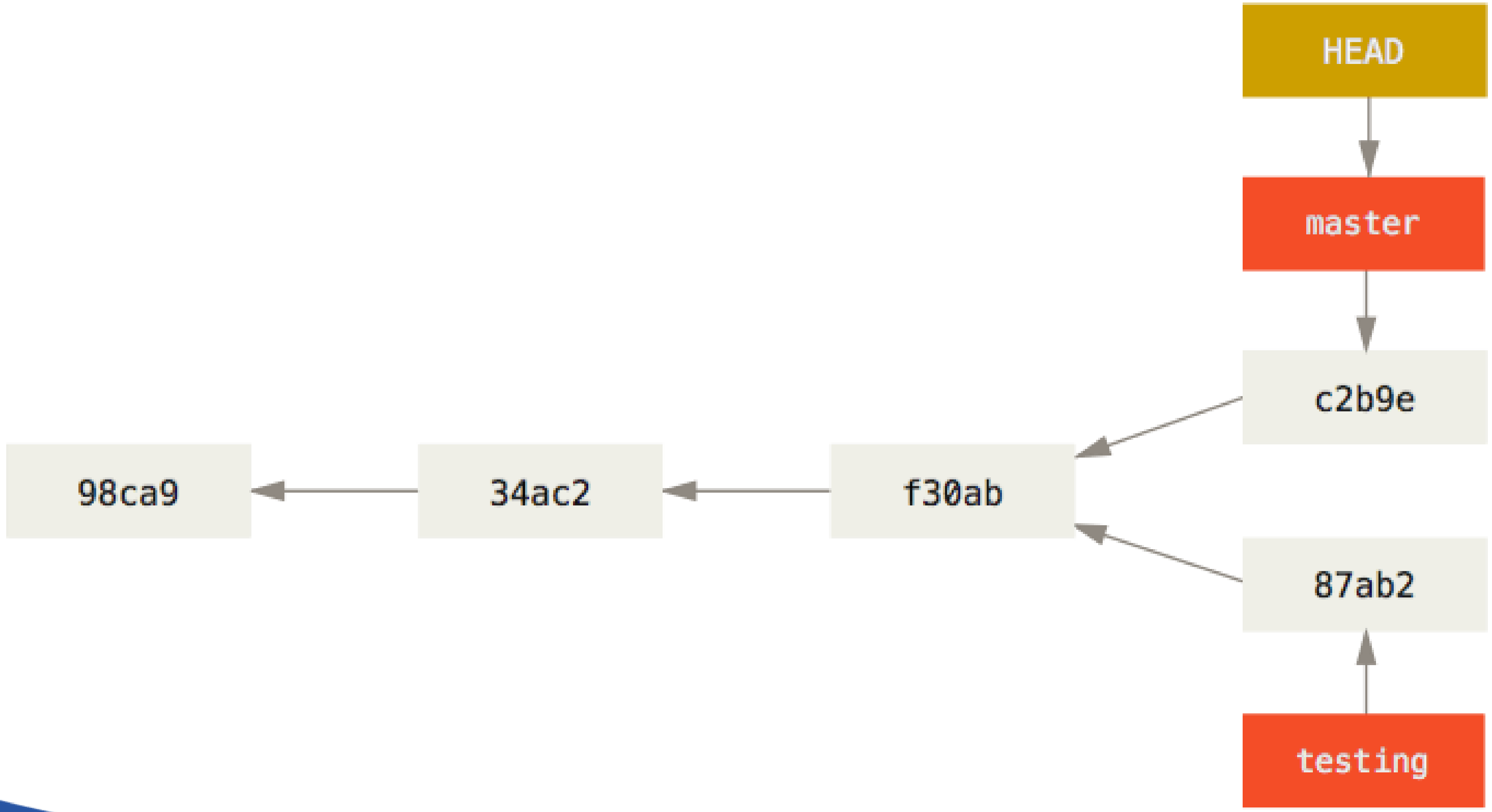
- We use git because it is popular, thus well-supported.
  - ⇒ A command-line (CLI) tool
  - ⇒ Many GUI applications wrap around the CLI (e.g., GitHub Desktop, GitKraken)
- This lesson: a quick tour of key git concepts.
  - ⇒ Many more resources available online—search and find some; identify the ones most helpful to you.
  - ⇒ Here we use diagrams from the Git Book (available in 19+ languages).

`git` concepts: commit

- single version of a set of files arranged in directories.
  - ⇒ Author, timestamp, files (‘blobs’), description.
  - ⇒ ID or ‘hash’ e.g. 3f2ca4130cab262cfac62c5a98dd2ebdeb424dc5.
    - We abbreviate with the first few characters: 3f2ca413
  - ⇒ Hash of a previous (‘parent’) commit.
  - ⇒ ‘Snapshots’ of each file.

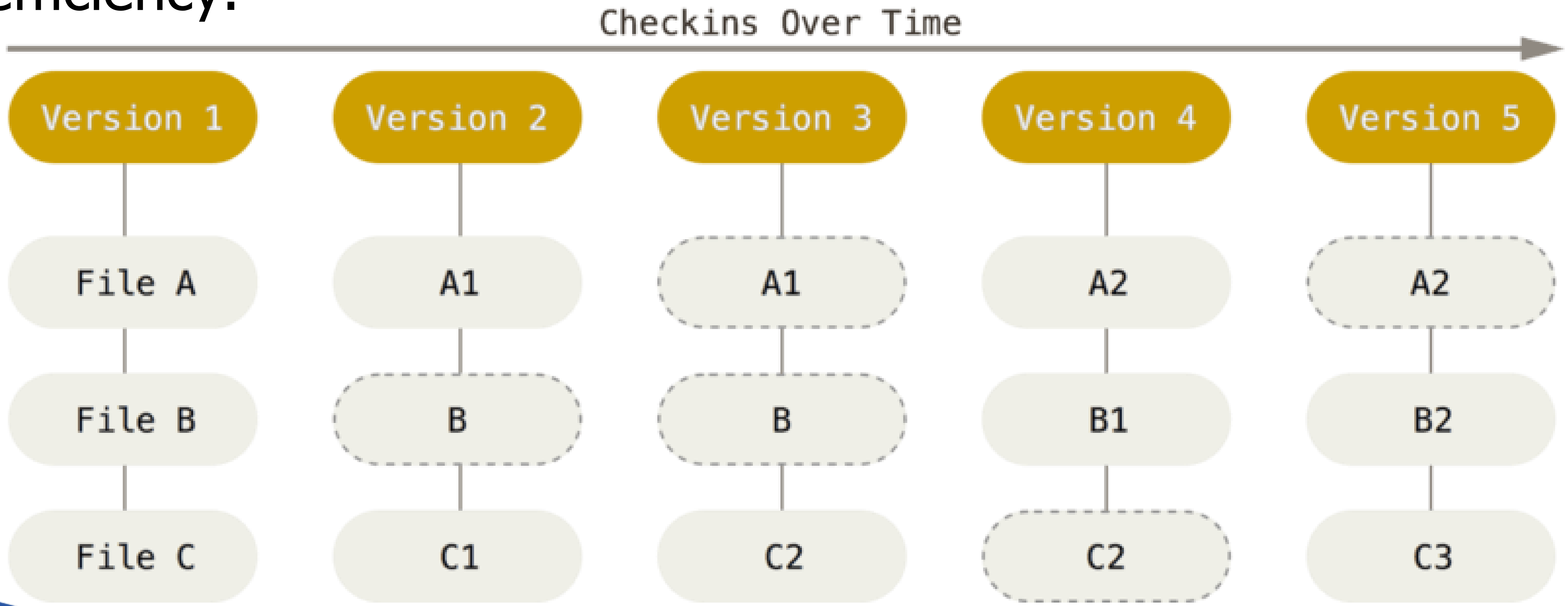
`git`: branch

A name for a particular commit and its ancestors:



`git`: branch

- Commits may share the same snapshot of a file → storage efficiency.



git concepts: diff

Used to express changes between two snapshots of a single file:

Original File

```
Shopping List

* Apples
* Oranges
* Salt
* Pepper
```

Modified File

```
Shopping List
for Friday

* Apples
* Oranges (1 dozen)
* Salt
```

Changes

```
 Shopping List
+for Friday

 * Apples
-* Oranges
+* Oranges (1 dozen)
 * Salt
-* Pepper
```

git doesn't store these internally, but understands & generates them.

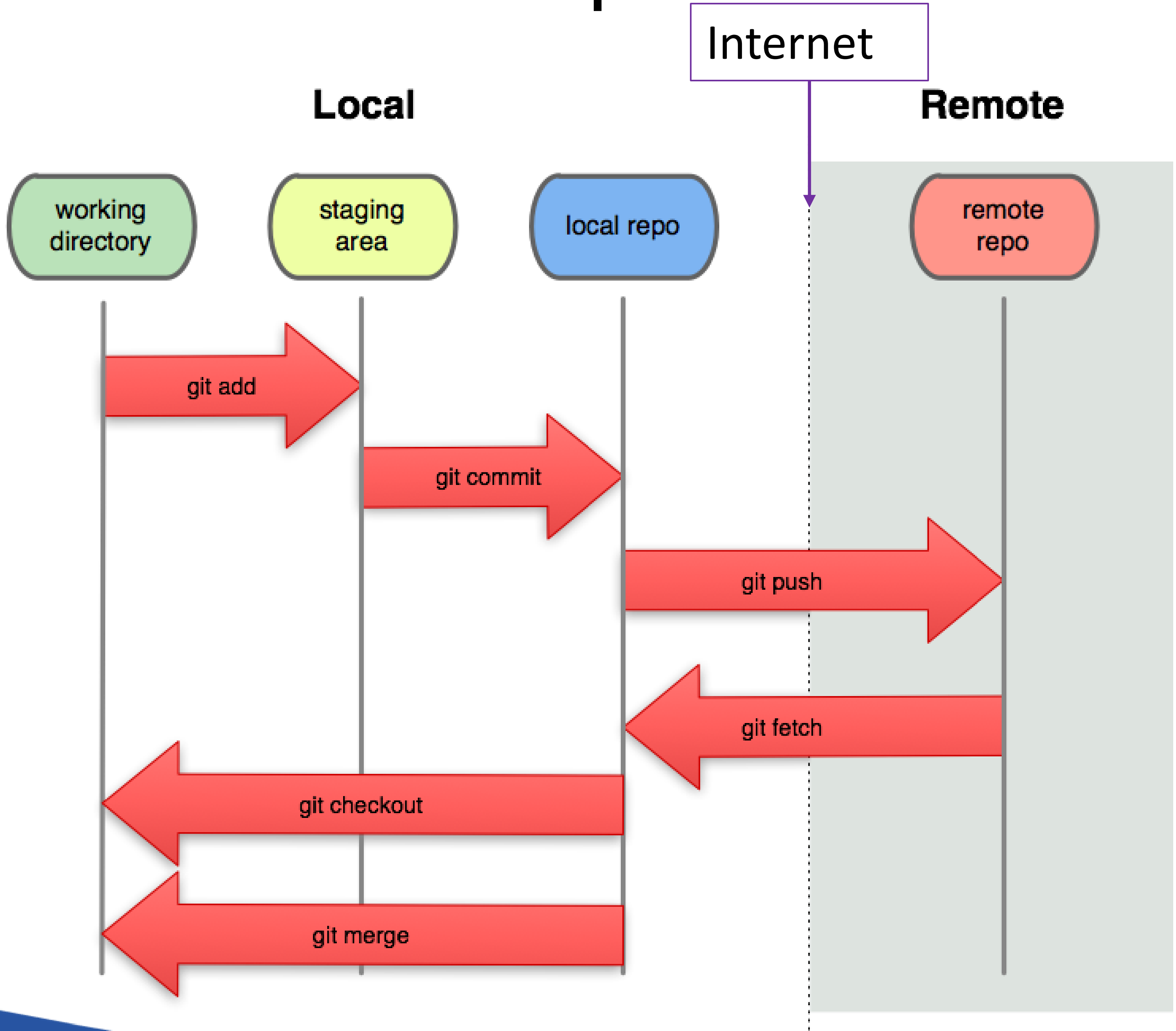
`git` concepts: tag

- A name applied to a certain commit.
- A branch can be extended by adding more commits to its head.
- A tag always stays in the same place.

`git`: repository (‘repo’)

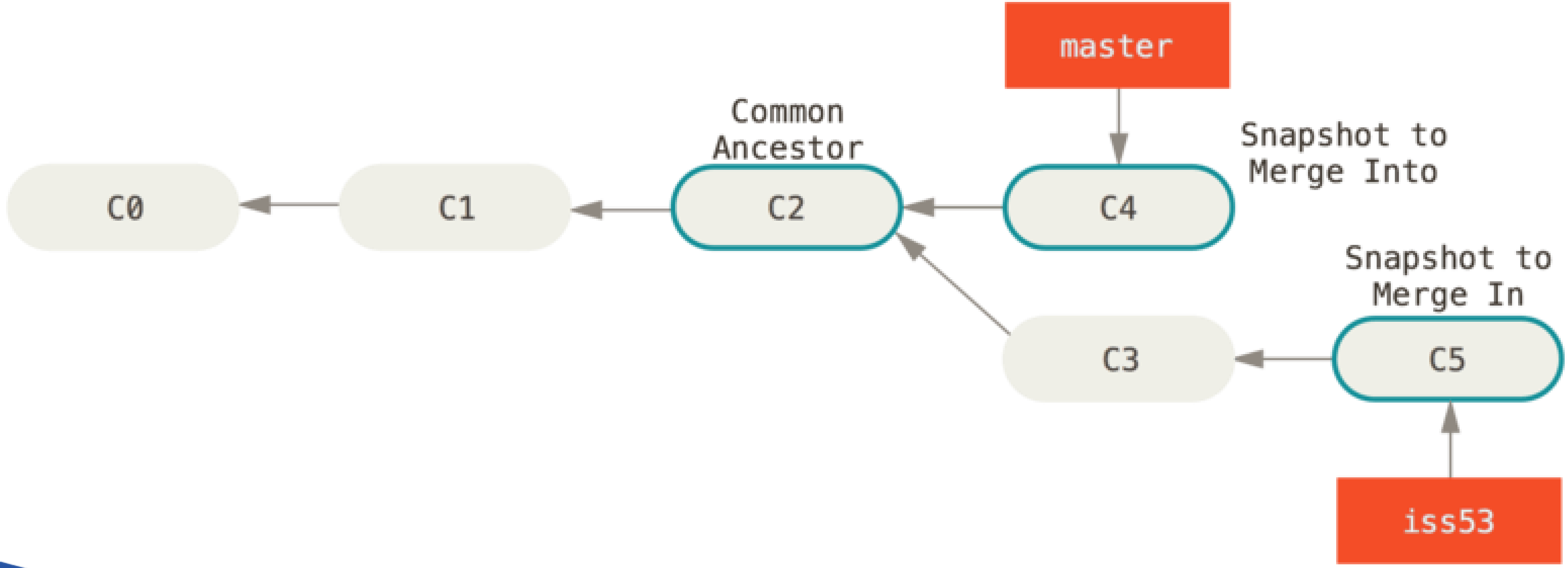
- A collection of commits, snapshots, and tags.

`git`: local and remote repositories



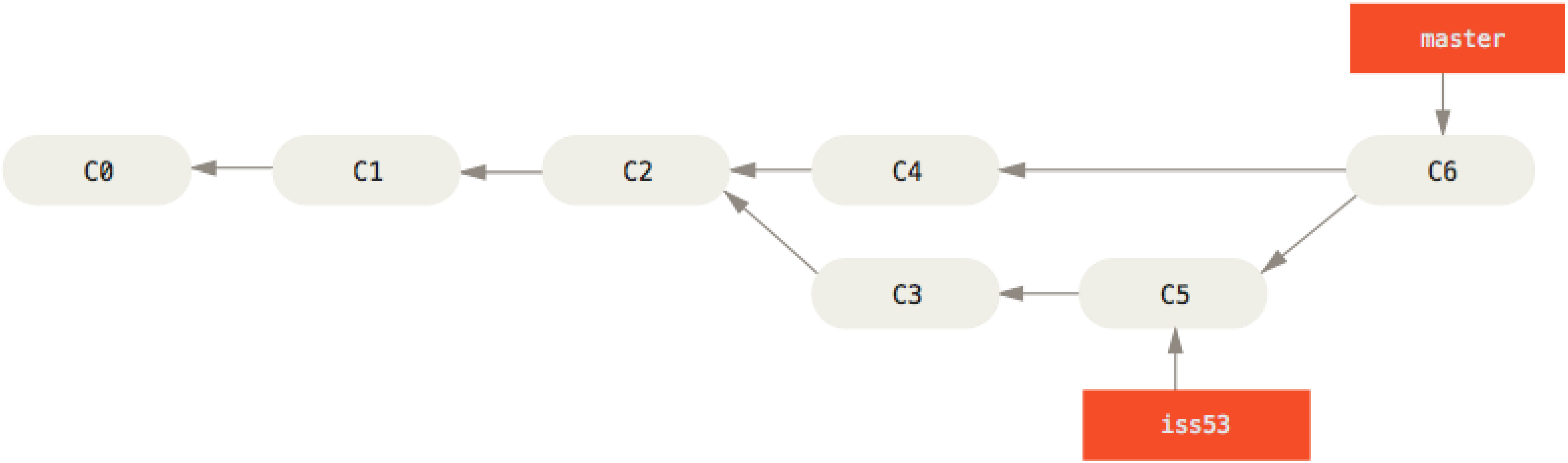
git: merge

- Combines two commits from different branches.



git: merge

- Creates a new commit.

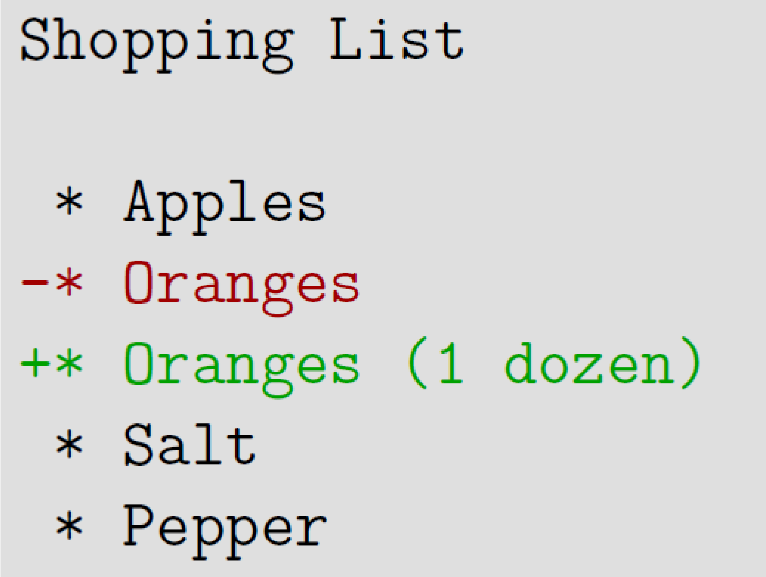


`git`: merge

- `git merge` automatically handles many tasks.
- For example, changes to the same file:
  - ⇒ `branch-a` has a commit that modified file.txt near the top.
  - ⇒ `branch-b` has a commit that modified file.txt near the bottom.
  - ⇒ git applies both changes because they are non-overlapping, producing a combined file.txt

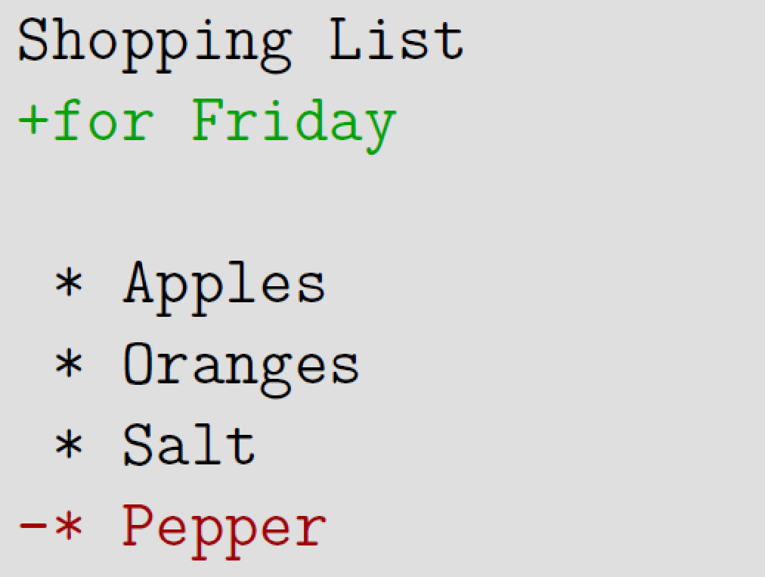
git: merge

Branch A changes



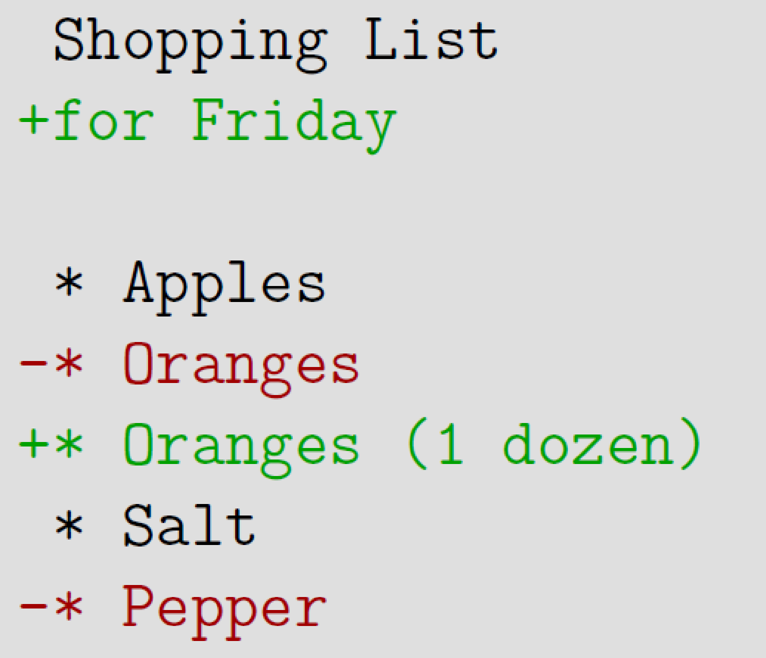
```
Shopping List

 * Apples
-* Oranges
+* Oranges (1 dozen)
 * Salt
 * Pepper
```

Branch B changes



```
Shopping List
+for Friday

 * Apples
 * Oranges
 * Salt
-* Pepper
```

Combined changes



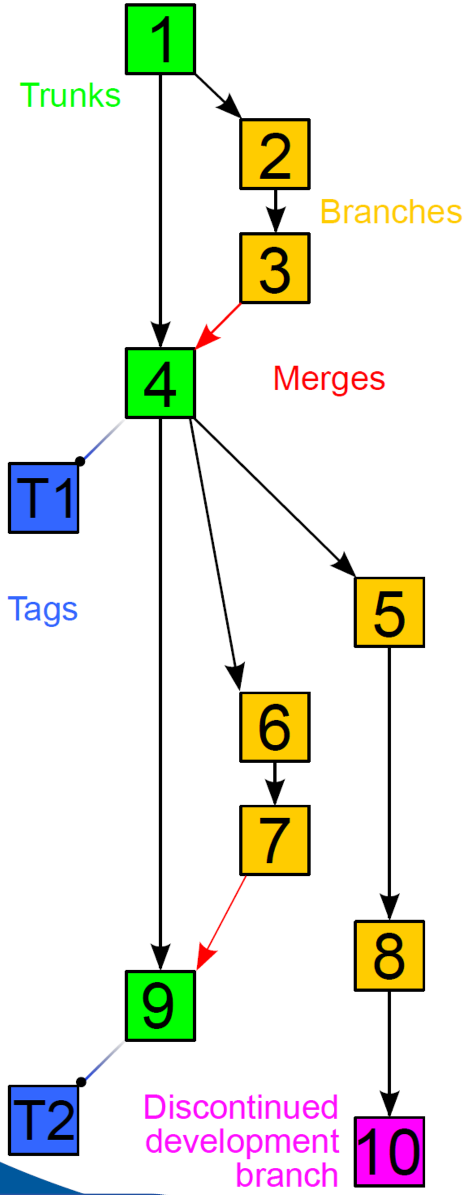
```
 Shopping List
+for Friday

 * Apples
-* Oranges
+* Oranges (1 dozen)
 * Salt
-* Pepper
```

→ keep files & directories neatly organized.

`git`: fetch/pull/push

- git can move commits between two repos in different places:
  - ⇒ Two folders/directories on the same computer.
  - ⇒ Two computers: yours vs. a colleague’s, or a server online.
- The other repo is called a remote. git helps you:
  - ⇒ Name and track multiple remotes related to the current repo.
  - ⇒ Associate a local branch with one branch on one remote.
- Operations
  - ⇒ fetch: copy commits, branches, tags from a remote repo to yours. Doesn’t change anything.
  - ⇒ pull: does three things
    1. Fetch a remote repo.
    2. Add new commits from the remote repo onto associated local branch.
    3. Fast-forward the pointer at the head of the local branch.
  - ⇒ push: pull, but in the opposite direction.

Another visualization



1, 2, …, 10 commits.

2, 3 a branch; work done in parallel. Others can get & use 1 while 2, 3 are developed.

4, 9 merge commits. The changes made in 2, 3 (or 6, 7) are combined with 1 (or 4) to produce the new revision 4 (or 9).

1, 4, 9 the 'master' branch

Chosen by the user to be the authoritative version of the code.

T1, T2 tags.

Collaborative development using GitHub

General Concepts

- VCS like git provide tools for managing versions of code.
- They do not:
  - ⇒ Require collaboration.
    You can use git in a single local repo without an Internet connection.
  - ⇒ Require that the files/code do anything, or be ‘correct’.
  - ⇒ Prescribe how or to what end we should use them.
- Software development comprises…
  - ⇒ the actions of conceiving, specifying, designing, programming, documenting, testing, and bug fixing…
  - ⇒ involved in creating and maintaining software.

General Concepts

- Collaborative development: when software development involves 2+ people embedded in 1+ organizations.
  - ⇒ Using a VCS can make this a lot easier, but…
  - ⇒ All involved must agree on how to use the VCS.
- To collaborate, we must communicate about code:
  - ⇒ "[code] used to do X for me, but now it doesn't."
  - ⇒ "[code] says it will do X, but instead does Y."
  - ⇒ "[Al's code] does X, [Bo's code] does Y, but Jo wants to do both."
  - ⇒ "We fixed Y by making [changes] to [code]."
  - ⇒ "I wrote [new code] and I want everyone to use it."
  - ⇒ "You should use [version] instead of [version]."

GitHub

- A (very) popular website.
- You (user) or a group (organization) can store git repos on their servers.
- More importantly, provides many tools for software development tasks
- (previous slide).
  - ⇒ These are tightly tied to specific git repos, branches, commits, and tags.
  - ⇒ They make it easy to use a certain workflow of software development.
  - ⇒ Understanding and using this workflow is a good basis for teams collaborating on software.

BUT (!)

- GitHub’s features are only higher-level tools, built on git.
- They suggest a certain workflow, but every set of collaborators must still decide whether and how to use the features, and what their use means.
- (!) below flags these decisions. For example:
  - ⇒ Alice and Bob both run into problems with Model X.
  - ⇒ Bob files a bug report (on GitHub) that doesn’t prompt any action.
  - ⇒ Alice doesn’t use GitHub at all. Her problem results in a new branch with many commits, lots of discussion, a quick merge into master, and a release—all via GitHub.
- Why did this happen?

GitHub workflow concepts: fork

- A repo that is created by copying another repo.
- Example:
  - ⇒ https://github.com/iiasa —IIASA organization.
  - ⇒ https://github.com/iiasa/message_ix —'main' repository for message_ix.
    - Can be made public or private.
    - View and push access can be controlled.
- https://github.com/volker-krey —user profile.
- https://github.com/volker-krey/message_ix —user's fork of message_ix.
- Useful for working on changes for private use, or isolating work before it is merged with the main repo.
- Can view all forks from a repo.

GitHub: release

- A git tag with title, description, and associated downloads.
- Example:

  ⇒ https://github.com/iiasa/message_ix/releases —all releases of message_ix.

GitHub: issue

- A discussion about some bug, planned feature, or other issue (!) related to a specific repo.
- Example: https://github.com/iiasa/message_ix/issues/244
  - ⇒ Identified by a number: iiasa/message_ix#244.
  - ⇒ Title and description from by the user who opened it; comments from others.
  - ⇒ Can be assigned to a particular user.
    (!) often the person responsible for fixing/addressing it.
  - ⇒ Can be associated with a label, milestone (later), or project (later).
  - ⇒ Status: open or closed. (!) Does 'closed' mean 'fixed'?
  - ⇒ https://github.com/iiasa/message_ix/issues —all issues for a repo. Search & filter tools.

GitHub: pull request (PR)

- A request to git merge one branch into another (the ‘base’).
- Example: https://github.com/iiasa/message_ix/pull/247
  - ⇒ Similar to issues: title, description, assignee(s), comments, label, milestone, project.
  - ⇒ Status: open, merged, or closed [without merging].
  - ⇒ Reviewer(s) — similar to assignees, 0+ other users (next slide).
  - ⇒ List of commits since the common ancestor.
  - ⇒ Collective diff for all changes introduced in the branch.
  - ⇒ Checks related to continuous integration tools (next lesson).
- Caution: a branch named iiasa:example is not the same as volker-krey:example!

GitHub: PR (continued)

- Pull requests can close a specific issue, e.g. by fixing a bug or adding a desired feature.
- Reviewers are requested, can view the commits and diff.
  - ⇒ Add comments on specific changed lines.
  - ⇒ Approve, request changes, or just comment.
- (!) Collaborators must decide how to use PRs/reviews:
  - ⇒ Are reviews required? How many?
  - ⇒ Who can review the code?
  - ⇒ Different reviewers for different parts of code/types of issues or PRs?
  - ⇒ Should the code itself contain certain things?
- https://github.com/iiasa/message_ix/pulls —all PRs for a repo.

GitHub: milestone

- A target for collecting issues and pull requests.
- Example: https://github.com/iiasa/message_ix/milestone/1
  - ⇒ Title and description.
  - ⇒ Status: open or closed.
  - ⇒ Can be assigned a target date.
  - ⇒ (!) What happens when the date passes?
  - ⇒ (!) Is a release created when the milestone is reached?

That was a lot of material…

- ..., but luckily there are cheat sheets and other online material.
- … and we’ll be using git and GitHub within this course so that you will have internalized some of the basic concepts by the end of the week.
- Hopefully this is not just useful for this course, but for your research in general.

A short introduction to GAMS

GAMS: General Algebraic Modeling System

- High-level modeling environment for mathematical programming and optimization (variables/equations)
- Language and interface to different solvers, e.g.
  - ⇒ Linear Programming (LP)
  - ⇒ Mixed Integer Programming (MIP)
  - ⇒ Non-Linear Programming (NLP)
- Flexible and easy to adapt
- Commercial system with demo mode
- License for teaching (~3 months, until January 2025)

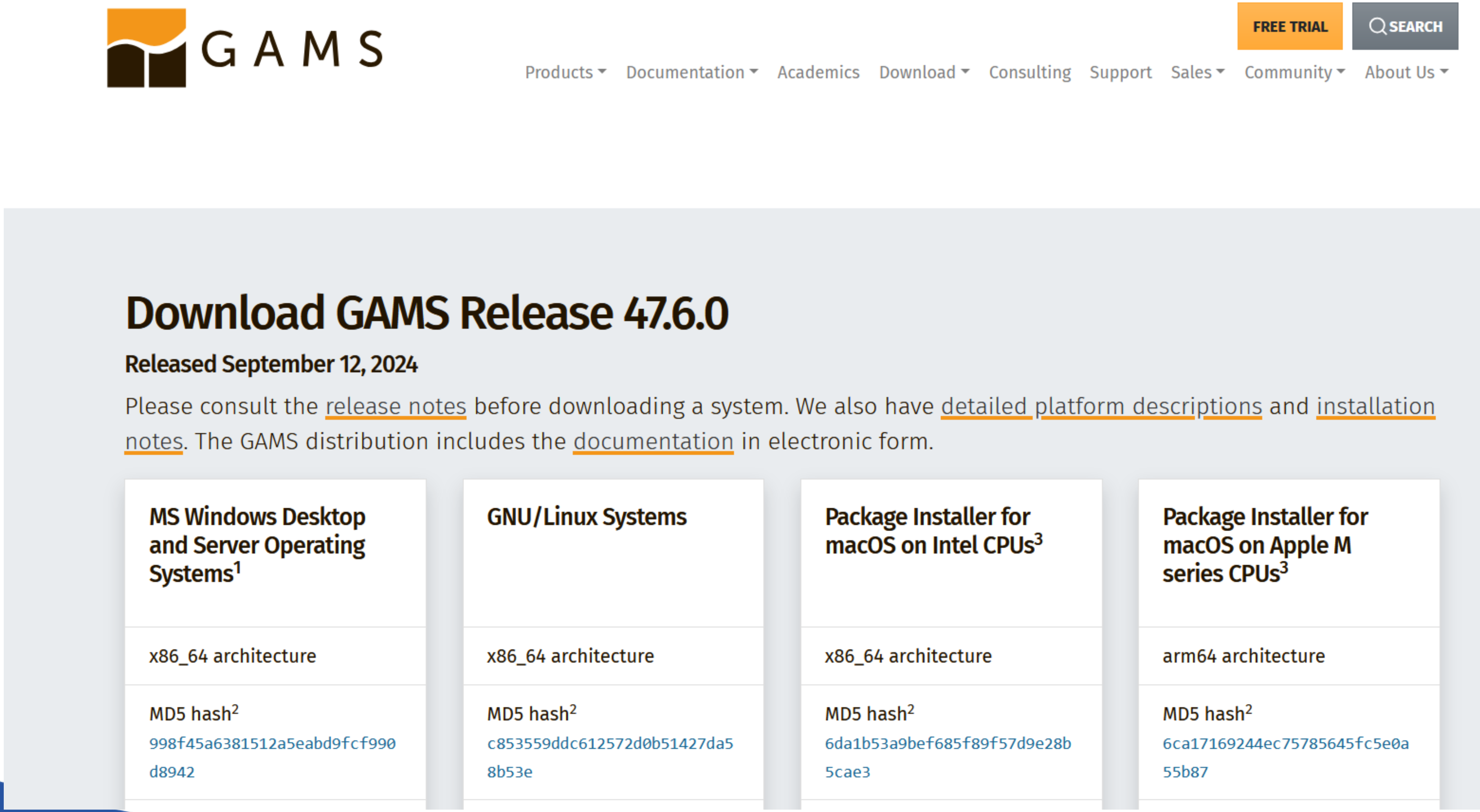
GAMS Installation

- GAMS Download under: http://gams.com/download/
- Installation via execution of platform-specific installer (e.g. `windows_x64_64.exe`)
- Start GAMS Studio as graphical user interface (or use older GAMS DIE)
- Use gamslice.txt teaching license provided for the course (or use an existing license for LP and NLP that you have access to)

Get started with a new model

- New Project in Menu File → New Project and choose base and working directory (usually directory with model code)
- Copy GAMS and open or create new model file in working directory (e.g., simple_model.gms, energy_model_world.gms)
- Run model via Menu File → Run (keyboard shortcut F9)

GAMS Website



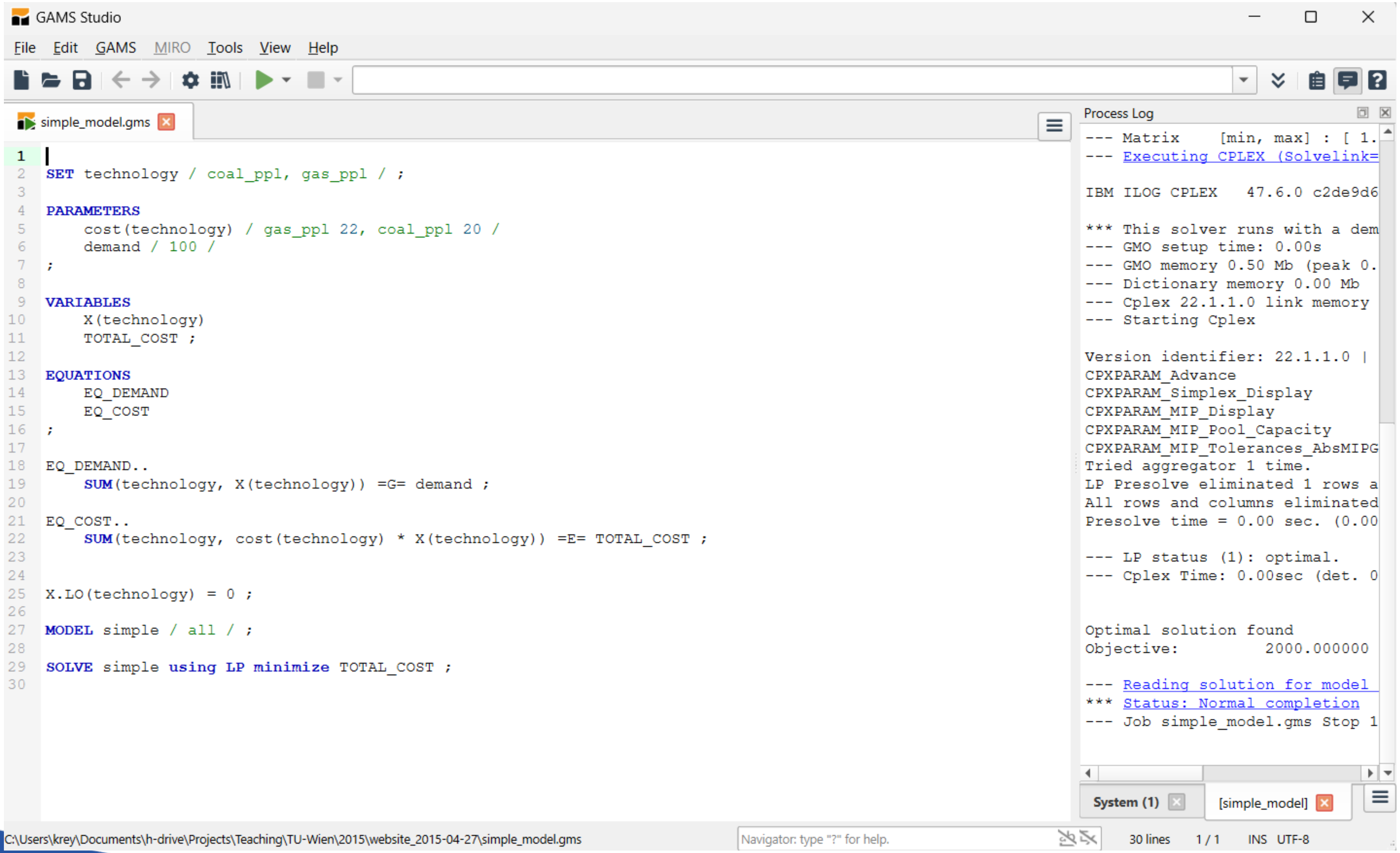
Download GAMS Release 47.6.0

Released September 12, 2024

Please consult the release notes before downloading a system. We also have detailed platform descriptions and installation notes. The GAMS distribution includes the documentation in electronic form.

MS Windows Desktop and Server Operating Systems[1]	GNU/Linux Systems	Package Installer for macOS on Intel CPUs[3]	Package Installer for macOS on Apple M series CPUs[3]
x86_64 architecture	x86_64 architecture	x86_64 architecture	arm64 architecture
MD5 hash[2] 998f45a6381512a5eabd9fcf990d8942	MD5 hash[2] c853559ddc612572d0b51427da58b53e	MD5 hash[2] 6da1b53a9bef685f89f57d9e28b5cae3	MD5 hash[2] 6ca17169244ec75785645fc5e0a55b87

GAMS Studio



```
SET technology / coal_ppl, gas_ppl / ;

PARAMETERS
    cost(technology) / gas_ppl 22, coal_ppl 20 /
    demand / 100 /
;

VARIABLES
    X(technology)
    TOTAL_COST ;

EQUATIONS
    EQ_DEMAND
    EQ_COST
;

EQ_DEMAND..
    SUM(technology, X(technology)) =G= demand ;

EQ_COST..
    SUM(technology, cost(technology) * X(technology)) =E= TOTAL_COST ;


X.LO(technology) = 0 ;

MODEL simple / all / ;

SOLVE simple using LP minimize TOTAL_COST ;
```

```
--- Matrix    [min, max] : [ 1.
--- Executing CPLEX (Solvelink=

IBM ILOG CPLEX   47.6.0 c2de9d6

*** This solver runs with a dem
--- GMO setup time: 0.00s
--- GMO memory 0.50 Mb (peak 0.
--- Dictionary memory 0.00 Mb
--- Cplex 22.1.1.0 link memory
--- Starting Cplex

Version identifier: 22.1.1.0 |
CPXPARAM_Advance
CPXPARAM_Simplex_Display
CPXPARAM_MIP_Display
CPXPARAM_MIP_Pool_Capacity
CPXPARAM_MIP_Tolerances_AbsMIPG
Tried aggregator 1 time.
LP Presolve eliminated 1 rows a
All rows and columns eliminated
Presolve time = 0.00 sec. (0.00

--- LP status (1): optimal.
--- Cplex Time: 0.00sec (det. 0


Optimal solution found
Objective:         2000.000000

--- Reading solution for model
*** Status: Normal completion
--- Job simple_model.gms Stop 1
```

C:\Users\krey\Documents\h-drive\Projects\Teaching\TU-Wien\2015\website_2015-04-27\simple_model.gms

Thank you very much for your attention!

Volker Krey
Integrated Assessment & Climate Change (IACC) Group
Energy, Climate & Environment (ECE) Program
International Institute for Applied Systems Analysis (IIASA)
Laxenburg, Austria
krey@iiasa.ac.at
www.iiasa.ac.at