Integrated Assessment Modelling

Volker Krey

NTNU course: Integrated Assessment Modelling (EP8900)

A few words about myself

- Diploma in physics (2002)
- PhD in mechanical engineering: energy systems modeling (2006)
- Research Scholar, Energy Program at IIASA (2007-2010)
- Broadened scope of work to integrated assessment modeling (incl. climate science, land use modeling, economics, …)
- Deputy Director – Energy Program at IIASA (2010-2020)
- Research Group Leader – Integrated Assessment & Climate Change Group at IIASA (since 2021)
- Adjunct Professor, Industrial Ecology Programme and Energy Transitions Initiative at NTNU (since 2018)
- Lead Author of IPCC WGIII SRREN (2009-2011), AR5 (2011-2014), AR6 (2019-2022)

Round of self-introduction

- What is your background?
- What do you hope to get out of this course?
- Do you have previous modeling experience?
- How much programming experience do you have?
- Do you have experience with version control systems and related tools (e.g., svn, git, GitHub)?
- Anything else?

Time: 15-20 minutes

An overview of this week

- Mix of lectures, hands-on session and group exercises.

- In general lectures in the mornings, hands-on sessions and group exercises in the afternoons.

- Feedback is welcome!

- Depending on pre-existing knowledge in group and feedback some flexibility in adjusting material and session formats.

Plan for this week

	Monday	Tuesday	Wednesday	Thursday	Friday
	14/02/2022	15/02/2022	16/02/2022	17/02/2022	18/02/2022
09:00-09:45	Self-introduction of participants Course introduction	Consultation on MESSAGEix installation and tutorials	Historical overview of IPCC scenarios, Representative Concentration Pathways (RCPs)	Consultation on hands-on session	Coupling IAMs to disciplinary models for assessing broader SDG implications
09:45-10:30	A short history of IAMs, different modeling paradigms and system boundaries		Shared Socio-economic Pathways (SSPs)		
10:30-10:50	Break		Break		Break
10:50-12:00	Good modeling practice and collaborative tools for model development		Integrating industrial ecology analysis into IAMs		Policy analysis: assessing current policies and NDCs
12:00-14:00	Lunch		Lunch		Lunch
14:00-16:00	Hands-on session: Installation of MESSAGEix and tutorial(s)		Hands-on session: MESSAGEix model development for assignments		Group work: wrapping up Planning ahead: seminar and assignments

Credits

- This lecture is based on ongoing research and capacity building activities.

- It builds on material from a number people, in particular: Shinichiro Fujimori, Matthew Gidden, Arnulf Grübler, Daniel Huppmann, Paul Kishimoto, Gunnar Luderer, David McCollum, Haewon McJeon, Clara Orthofer, Shonali Pachauri, Simon Parkinson, Peter Rafaj, Keywan Riahi, Heleen van Soest, Charlie Wilson, Behnam Zakeri

GitHub repository

- Create a GitHub user account: https://github.com/

- E-Mail account name to krey@iiasa.ac.at

- Go to ntnu_iam_2022 repository: https://github.com/iiasa/ntnu_iam_2022

- Create a *fork* of the repository under your username

Introduction to IAMs

What is integrated assessment modeling?

"integrated assessment is an attempt to combine information, analysis and insights from the physical and social sciences to address the nature of climate change and to develop possible policy responses to it"

John P. Weyant

Climatic Change 95,

p. 317–323, 2009

Emergence of IAMs

- Formally modelled integrated assessment studies trace their inspiration, if not their precise methods, to the global models of the 1970s (Meadows et al. 1972, Mesarovic and Pestel 1974).

- Formal integrated assessment models emerged in the late 1970s from earlier economic and technical models of energy policy (e.g., Nordhaus 1979, Hafele et al. 1981, Nordhaus and Yohe 1983, Edmonds and Reilly 1985).

- The first integrated assessment model to extend fully from emissions to impacts did not address climate change but the more analytically tractable issue of acid rain (RAINS model developed at IIASA in the early 1980s, Alcamo, Shaw and Hordijk 1990).

- IPCC First Assessment Report (IPCC 1990) relied on Atmospheric Stabilization Framework (Lashof and Tirpak 1989) and IMAGE 1.0 (Rotmans 1990).

- Landmark of the maturation of IAMs of climate change was the first conference to assess activity in the field (Nakicenovic et al. 1994).

Source: IPCC SAR, WGIII, Chapter 10, Box 10.1

How do IAMs represent …?

It depends … IAMs come in different flavors

- Cost-Benefit vs. Process-based IAMs

Process-based IAMs

- "Roots":
  Energy systems vs. Macro-economic models
- Solution concept:
  Optimization vs. Simulation
- Equilibrium concept: Partial vs. General
- Solution horizon: Intertemporal vs. Recursive-dynamic
- Model design: Monolithic vs. Modular

Two Kinds of Integrated Assessment Models

- Policy Optimization Models (aka cost-benefit models)
  - $\Rightarrow$ Focused on finding optimal level of emissions
  - $\Rightarrow$ Usually include impacts at the aggregate level
- Policy Evaluation Models (aka process-based models)
  - $\Rightarrow$ Focused on simulating effects of policies
  - $\Rightarrow$ Usually much more detailed representation of processes and possibly impacts
  - $\Rightarrow$ Can be run backwards – tolerable windows approach

Courtesy of John Weyant (Stanford)

"Noble Prize" in Economics 2018

Climate change – Nordhaus' findings deal with interactions between society and nature. Nordhaus decided to work on this topic in the 1970s, as scientists had become increasingly worried about the combustion of fossil fuel resulting in a warmer climate. In the mid-1990s, he became the first person to create an *integrated assessment model,* i.e. a quantitative model that describes the global interplay between the economy and the climate. His model integrates theories and empirical results from physics, chemistry and economics. Nordhaus' model is now widely spread and is used to simulate how the economy and the climate co-evolve. It is used to examine the consequences of climate policy interventions, for example carbon taxes.
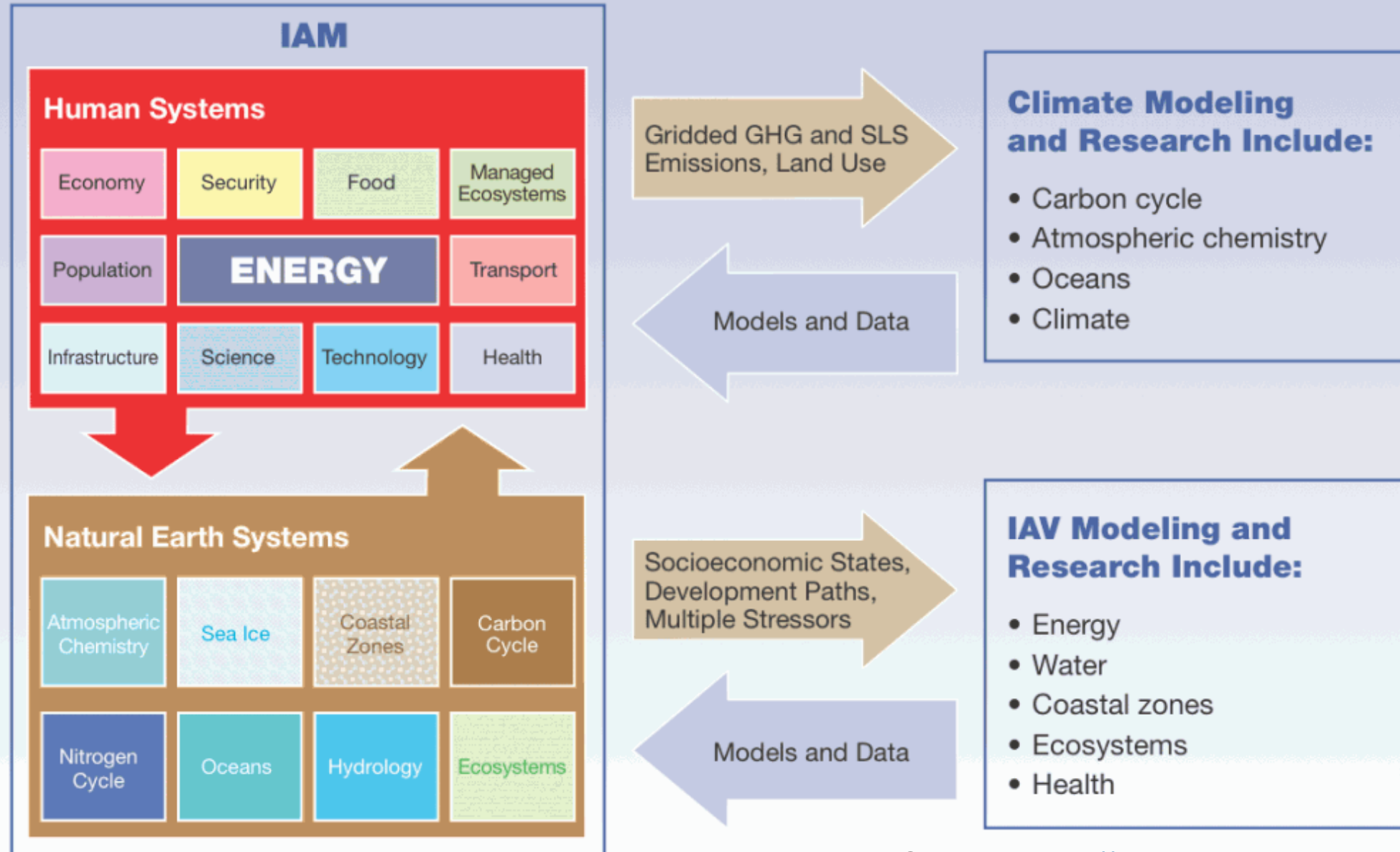

William D. Nordhaus

Cost-Benefit IAMs

Approach: Balancing the Costs of Controlling Carbon Emissions

Courtesy of John Weyant (Stanford)
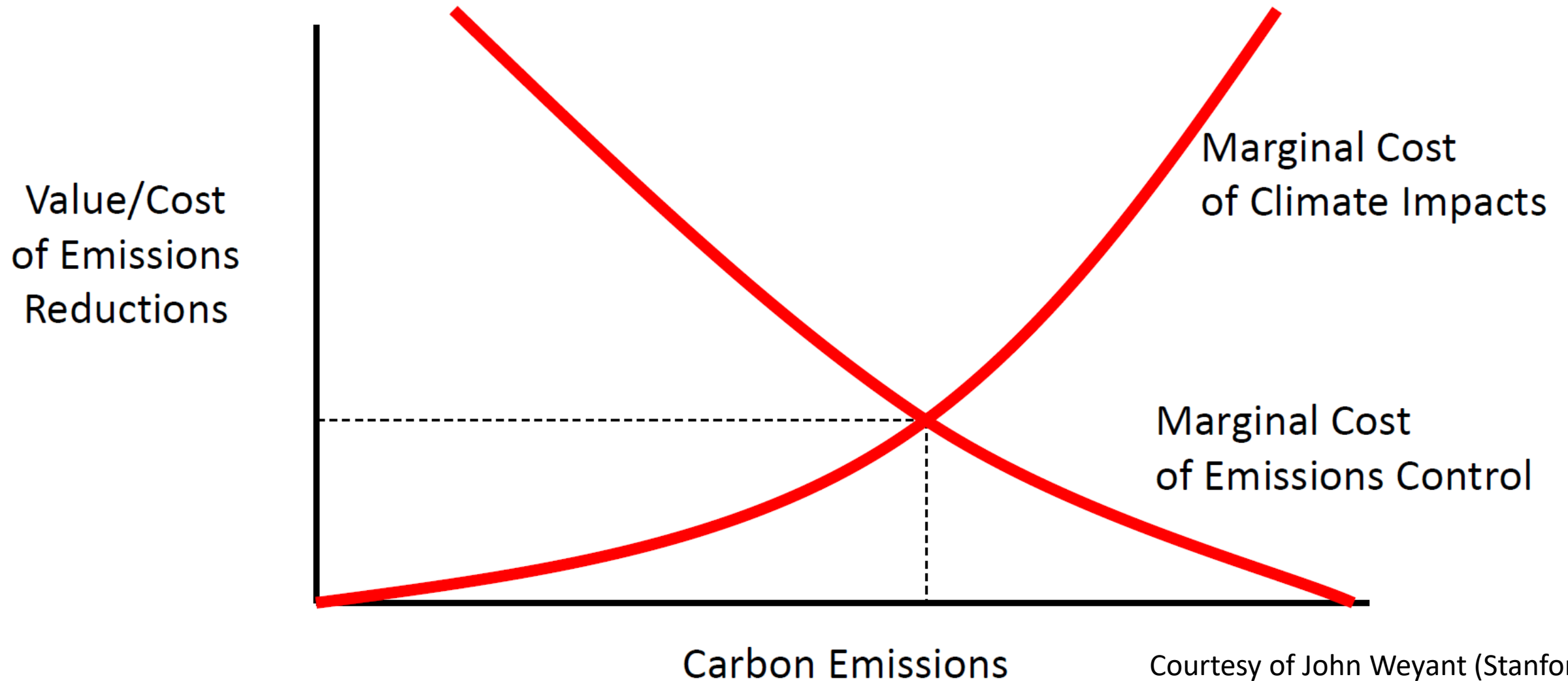
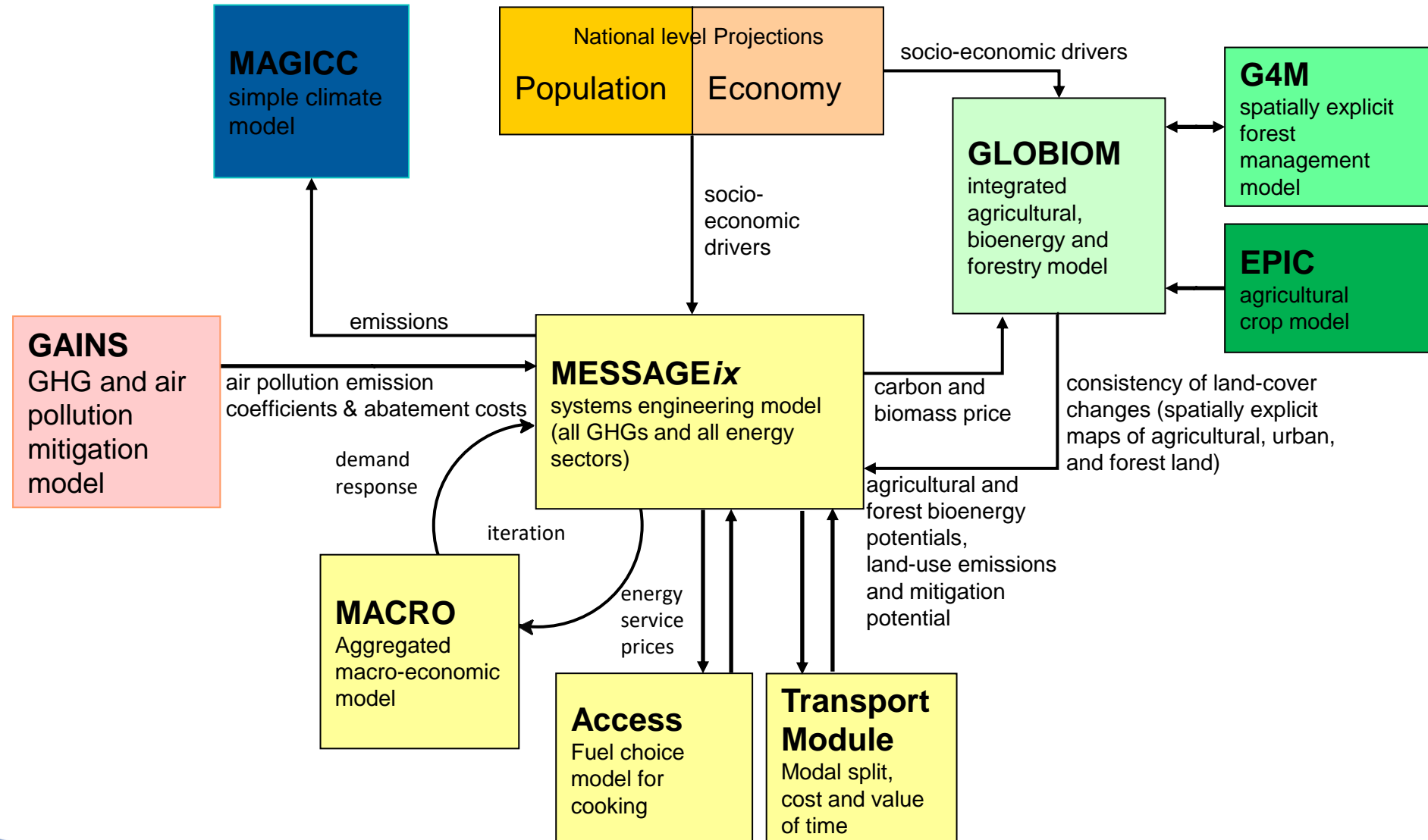# Generic Types of process-based IAMs

- Process engineering (aka activity analysis) models
  - ⇒ Individual technologies represented
  - ⇒ Need to add in market and economy wide effects
- Energy market (aka partial equilibrium) models
  - ⇒ Bring in energy market feedbacks
  - ⇒ Weaker on technology & economy
- "General equilibrium" models
  - ⇒ Bring in economic growth & economy-wide feedbacks
  - ⇒ Optimal growth (e.g., REMIND) and/or CGE (e.g., EPPA) variants
  - ⇒ Weaker on energy markets and technology
- Most models are hybrids of above types

Courtesy of John Weyant (Stanford)

IAMs Draw from and Serve Other Climate Science Research

IAM

Human Systems
- Economy
- Security
- Food
- Managed Ecosystems
- Population
- **ENERGY**
- Transport
- Infrastructure
- Science
- Technology
- Health

Natural Earth Systems
- Atmospheric Chemistry
- Sea Ice
- Coastal Zones
- Carbon Cycle
- Nitrogen Cycle
- Oceans
- Hydrology
- Ecosystems

Gridded GHG and SLS Emissions, Land Use →

Models and Data ←

Climate Modeling and Research Include:
- Carbon cycle
- Atmospheric chemistry
- Oceans
- Climate

Socioeconomic States, Development Paths, Multiple Stressors →

Models and Data ←

IAV Modeling and Research Include:
- Energy
- Water
- Coastal zones
- Ecosystems
- Health

Source: http://iamconsortium.org/

Example: IIASA Integrated Assessment Framework

Examples of IAM research questions

- How much do current policies and the NDCs achieve on the way to limit temperature change to 1.5 and 2°C?

- What are investment needs to limit temperature rise to 1.5 and 2°C?

- What are implications of climate policy to achieve the 1.5 and 2°C targets for SDGs?

- How much can behavioral change contribute towards mitigation (e.g., diets, transport)?

Strengths and Limitations

Strengths

- Globally and sectorally comprehensive analysis
- Interlinkages between sectors, regions (incl. trade in some commodities), human and natural systems

Limitations

- Technology focus – integration of behavioral aspects emergent
- Supply side bias – demand side typically less well represented
- Representation of national/sectoral barriers
- Limited spatial and temporal resolution
- Sectoral detail, e.g., IAMs are not power system models

"The purpose of computing is insights, not numbers."

Richard W. Hamming

Numerical Methods for Scientists and Engineers

McGraw-Hill, 1962

"*Essentially, all models are wrong, but some are useful.*"

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?
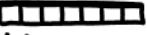
Your closest collaborator is you six months ago, but you don't reply to emails.

Productivity vs. Best Practices Overhead
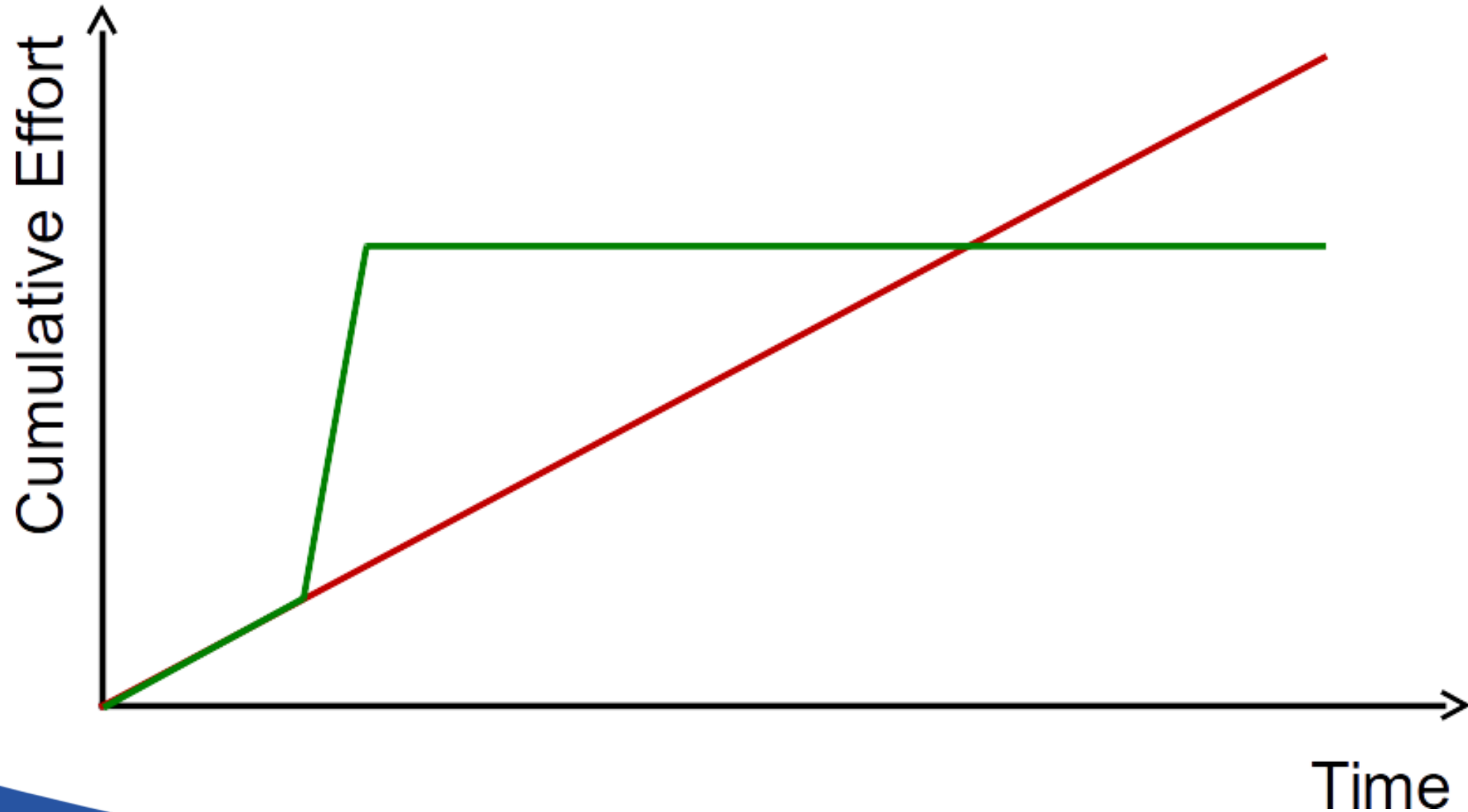
Time allocation for increasing efficiency through automation

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 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

HOW MUCH TIME YOU SHAVE OFF

Version control using git & GitHub

Based on a lectures by Paul Kishimoto and Matthew Gidden

Version Control
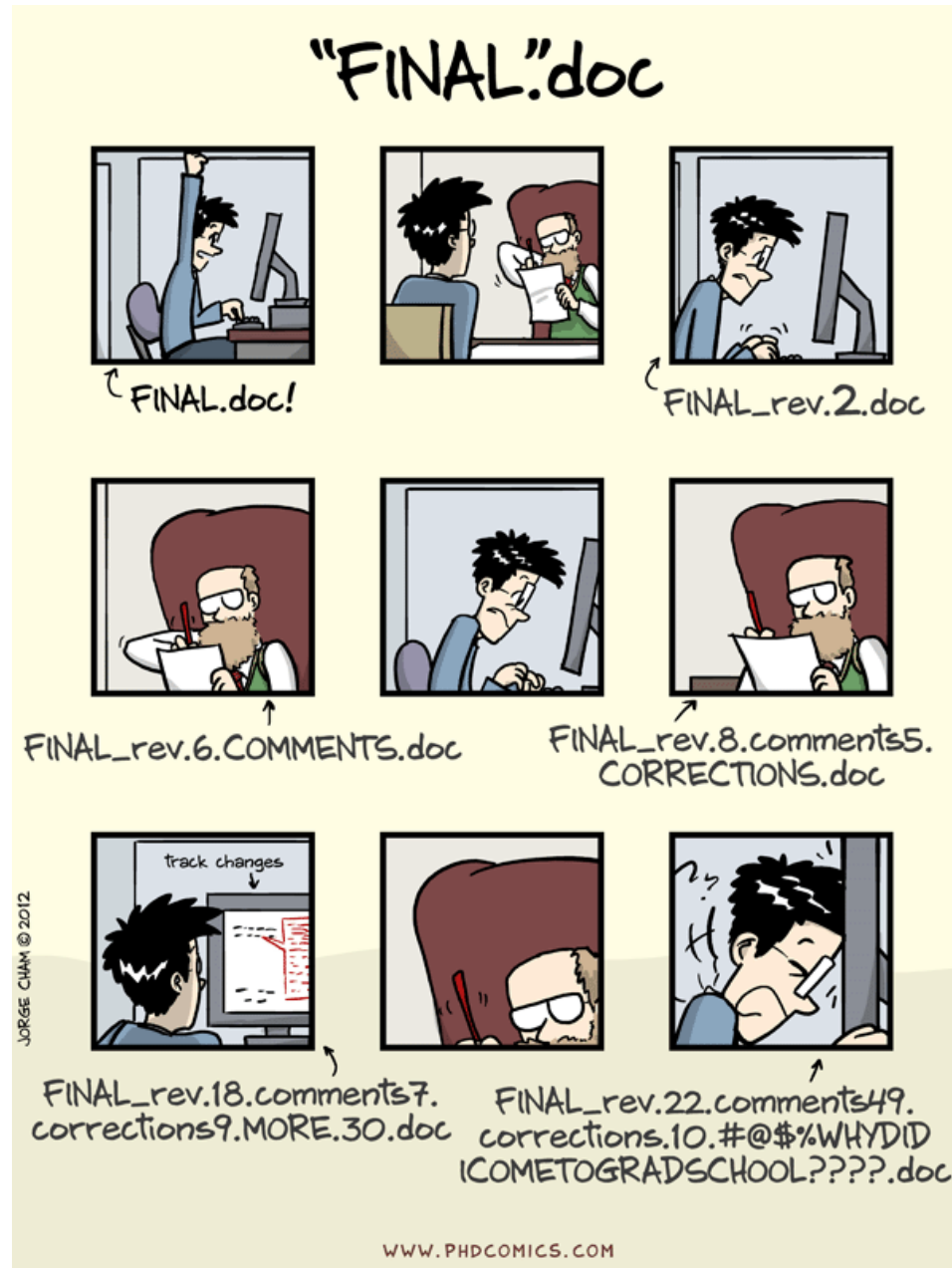
Version Control

- Save your work in "units"
- Lower cognitive load
- Easier to find bugs
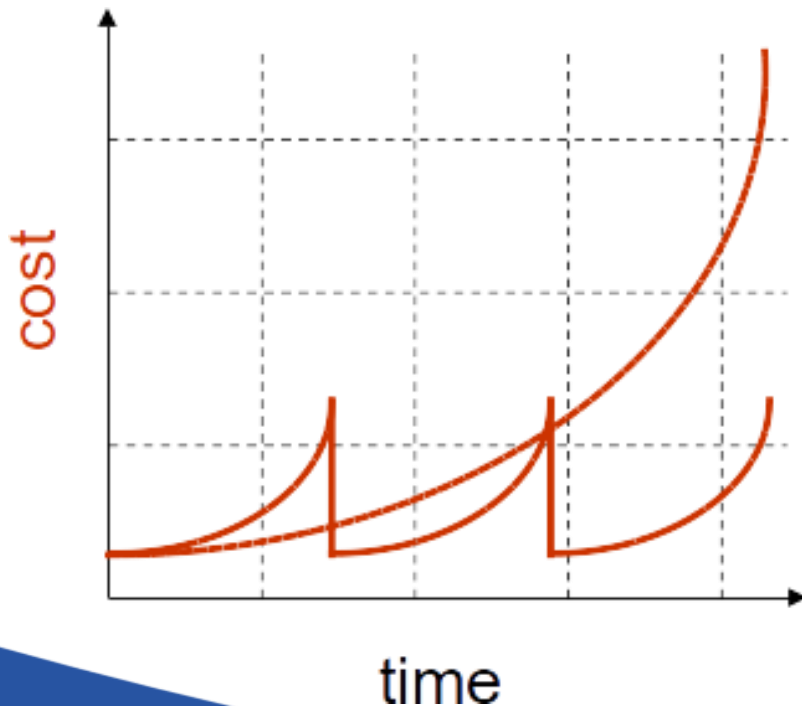
Two primary choices:
1. Centralized
   ⇒ SVN, etc.
2. Decentralized
   ⇒ git, etc.

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, Atom editor, GitKraken)
- This lesson: a quick tour of key git concepts.
  - ⇒ I 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:
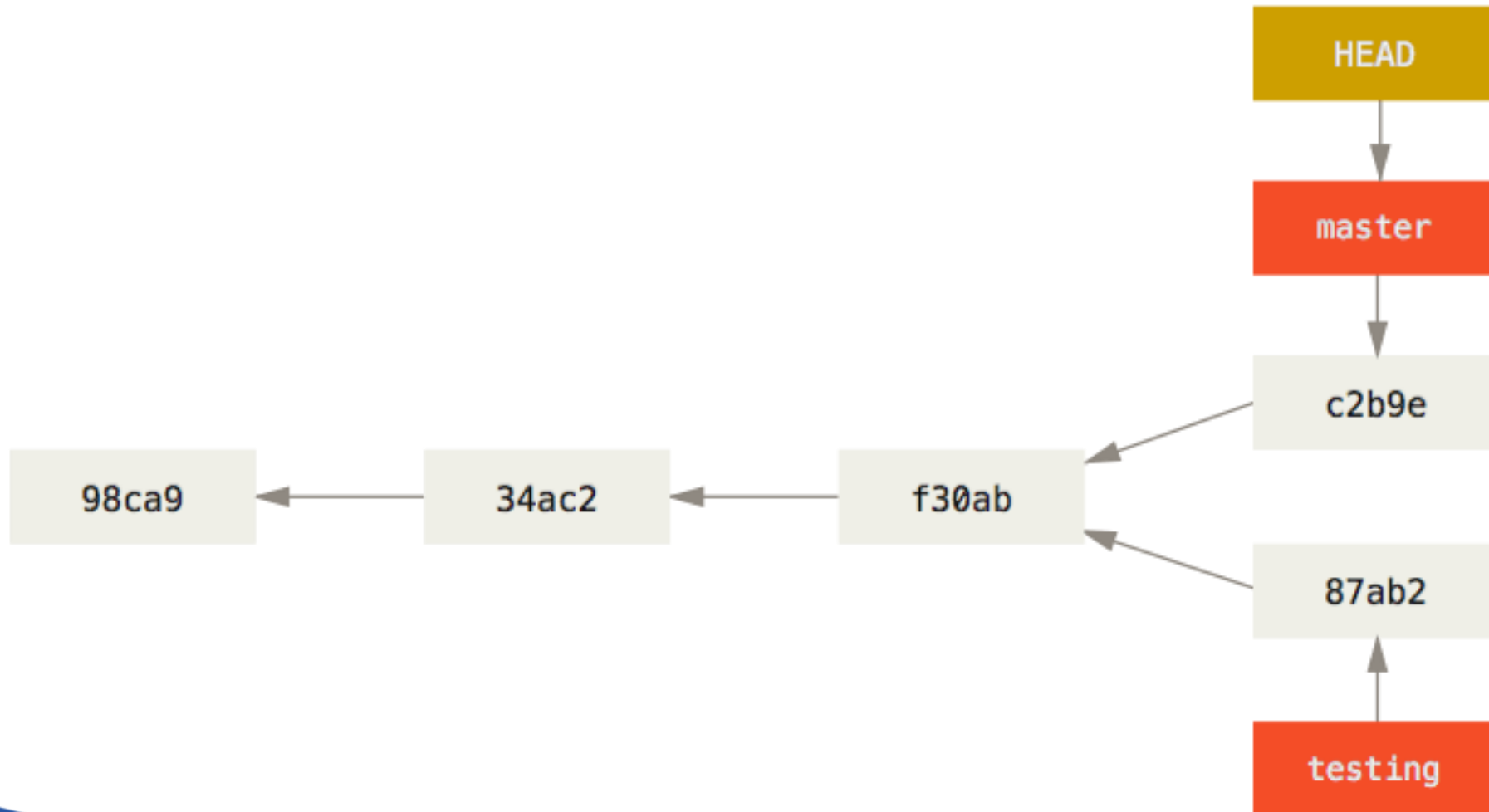
`git`: branch

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

Checkins Over Time
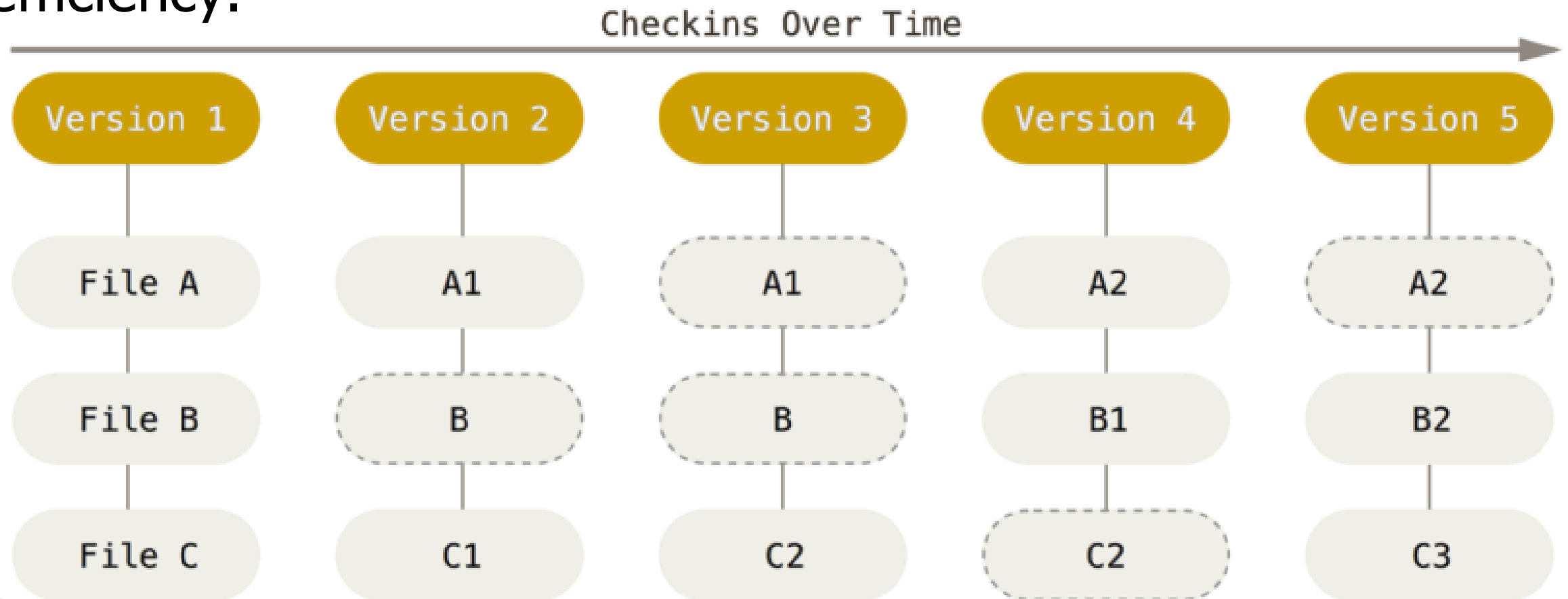
`git` concepts: diff

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

| Original File | Modified File | Changes |

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
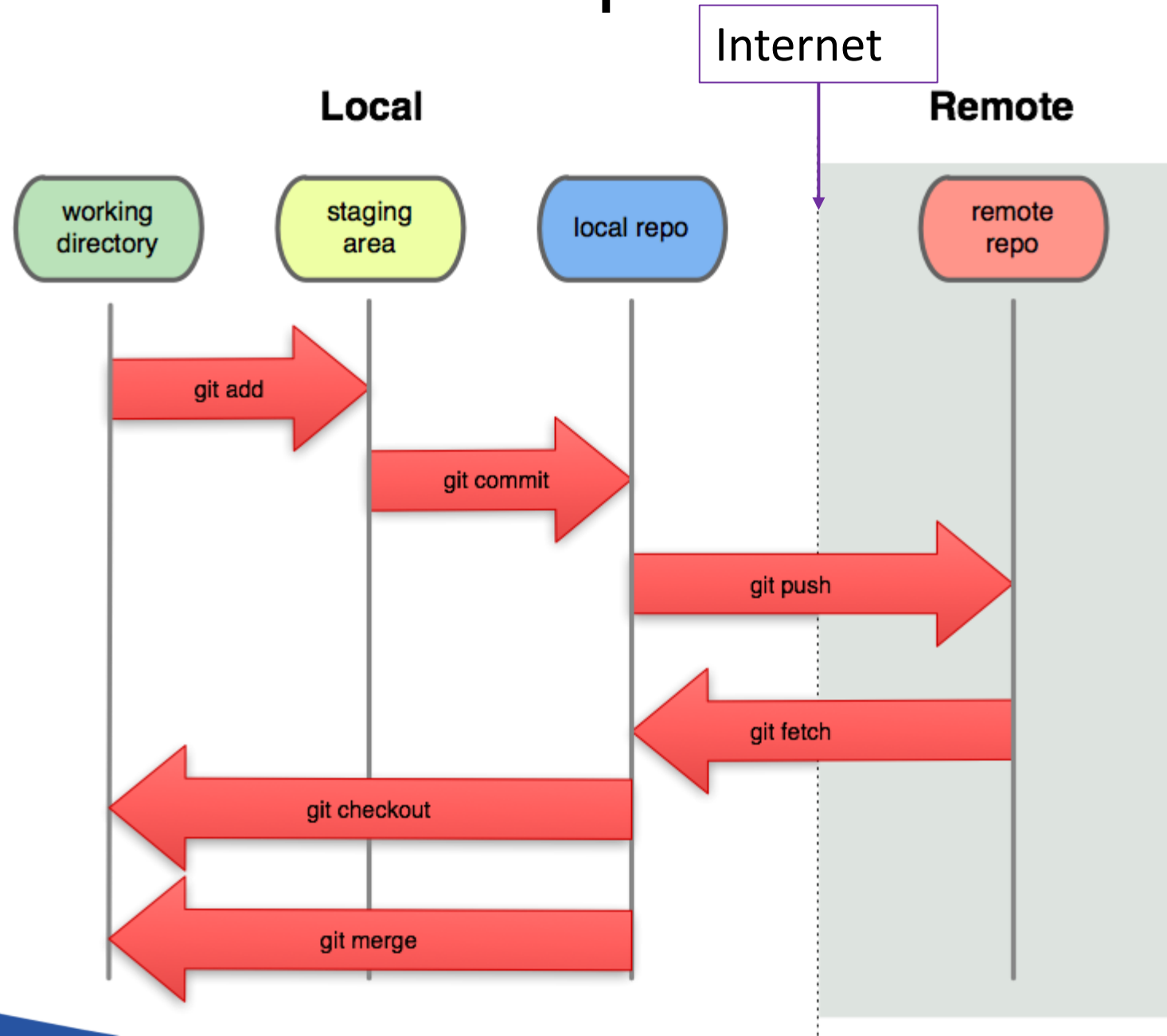
`git`: merge

- Combines two commits from different branches.
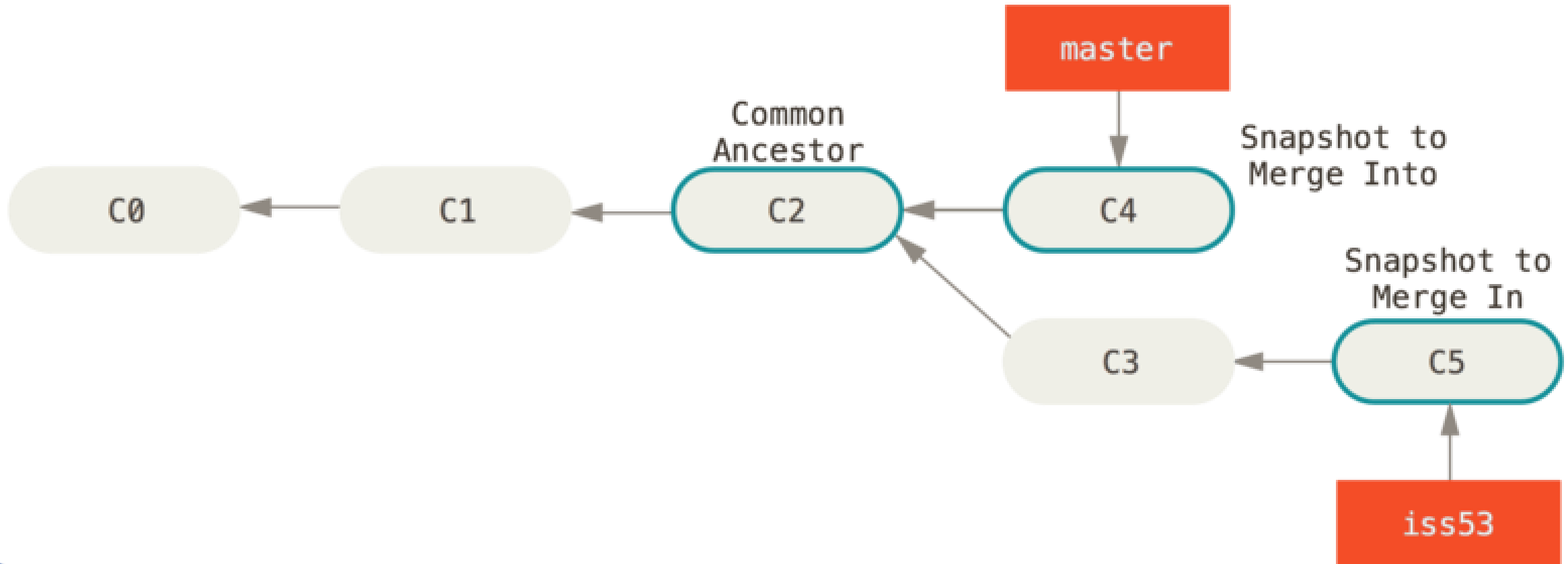
`git`: merge

- Creates a new commit.
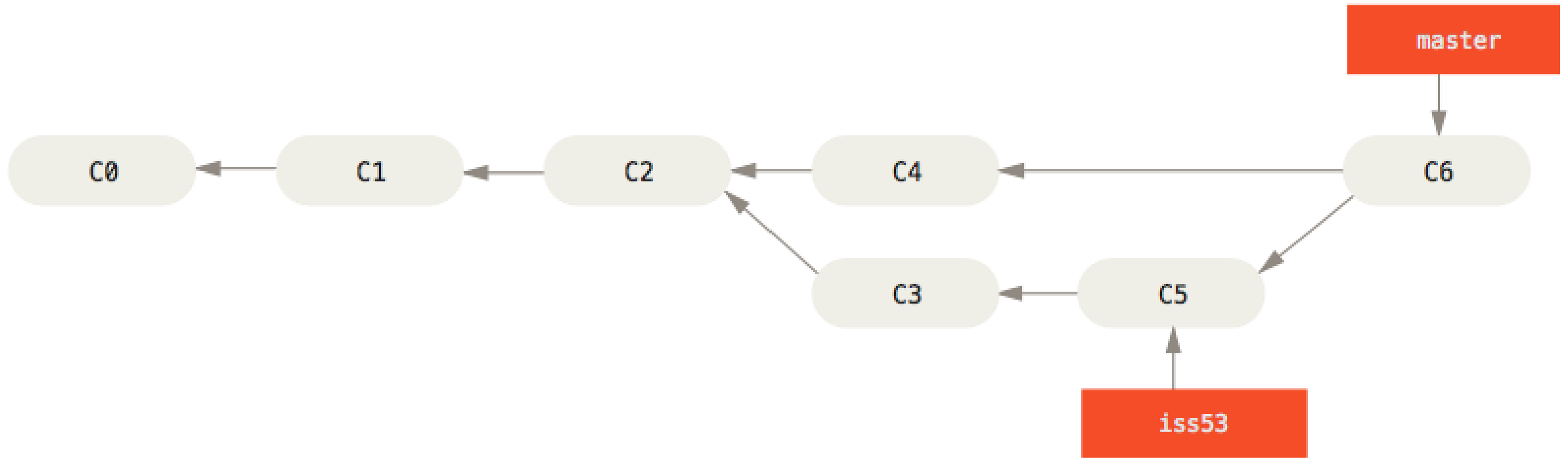
`git`: merge

- `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

Trunks

Branches

Merges

Tags

Discontinued development branch
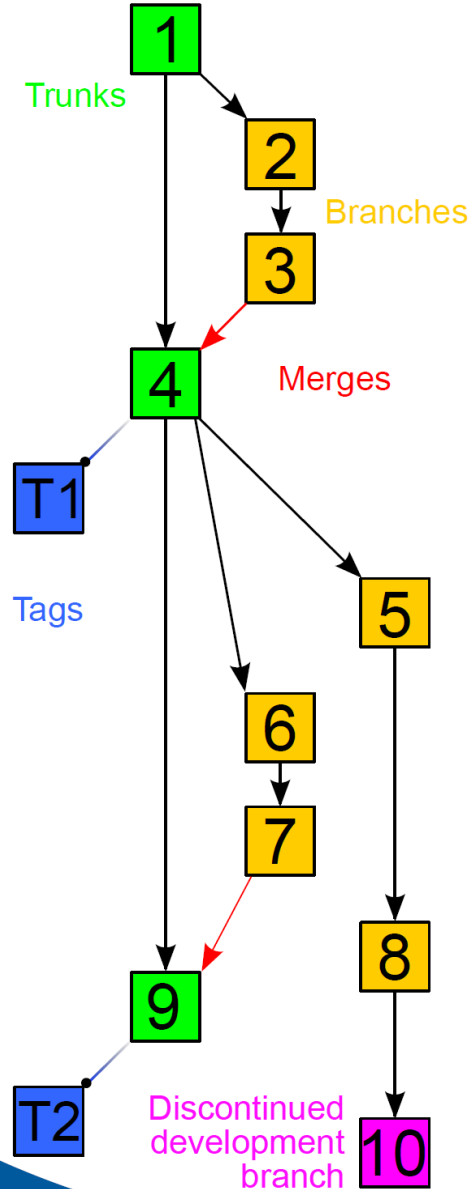
1, 2, …, 10	commits.
2, 3	a branch; work done in parallel. Others can get & use 1 while 2, 3 are developed.
4, 9	merge commits. The changes made in 2, 3 (or 6, 7) are combined with 1 (or 4) to produce the new revision 4 (or 9).
1, 4, 9	the 'master' branch
	Chosen by the user to be the authoritative version 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 excursion:
test-driven development and continuous integration

Why tests?

- Software tests ensure that software meets quality standards.
- Different kinds of tests ensure that…
  ⇒ the code works as intended—in part and in whole;
  ⇒ improvements do not cause regressions—breakage of existing features;
  ⇒ speed, memory/storage use, and other performance metrics are within desired limits; and/or
  ⇒ the software can be installed and used in its intended environment(s).
- Testing can be done manually (following a list of instructions), but is most commonly additional code that:
  1. Operates the target code in a certain way.
  2. Checks outputs or performance against expected outcomes.

Types of tests

- Unit tests of specific functions or lines—small pieces of code.
- Integration tests of the interactions between lower-level components.
- System tests of the completely integrated system.
- Test-driven development:
  1. Write tests first, as a way of saying:
     - "The code we write should do this."
     - "The bug reported by Person A represents a failure to do this."
  2. Write or edit code until the tests pass.
  3. Iterate as needed.

Test coverage

- A metric that determines whether part or all of the software is tested.
- Often measured as a number:
  (number of lines of tested code) / (total lines of code)
- Some common targets:
  ⇒ Code changes/additions must not decrease overall test coverage.
  ⇒ New additions must have 100% test coverage.
  ⇒ Test coverage must be 100%.
- Not the only important concept! Code may be 100% covered by unit-level tests, yet still fail to integrate or work correctly at higher, system level(s).

Purpose of Continuous Integration (CI)

- "integration tests" confirm that lower-level components interact properly.

- Continuous integration: frequent execution of integration and other tests.

  ⇒ For atomic changes, i.e. individual commits, or according to a schedule, e.g. nightly.

  ⇒ Performed by an automated system using test code and other configuration.

Why use CI?

- To avoid rework.
  - ⇒ Running tests only at the end of a process of developing new code may turn up unexpected bugs or incompatibilities.
  - ⇒ This results in rework: re-writing code again to address these problems.
- To be be robust to external conditions.
  - ⇒ Code that relies on e.g. an external web service or data source may be broken if that service/source changes.
  - ⇒ CI makes these issues visible immediately, so they may be fixed.
- To enforce quality without human intervention.
  - ⇒ This reduces the workload on pull request reviewers.

Examples of CI tools

- Travis
  - ⇒ Runs the entire MESSAGEix test suite on Linux and macOS (Windows experimental)
- Appveyor
  - ⇒ Runs the entire MESSAGEix test suite on Windows.
- Read The Docs (RTD)
  - ⇒ Builds (and hosts) the MESSAGEix documentation.

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