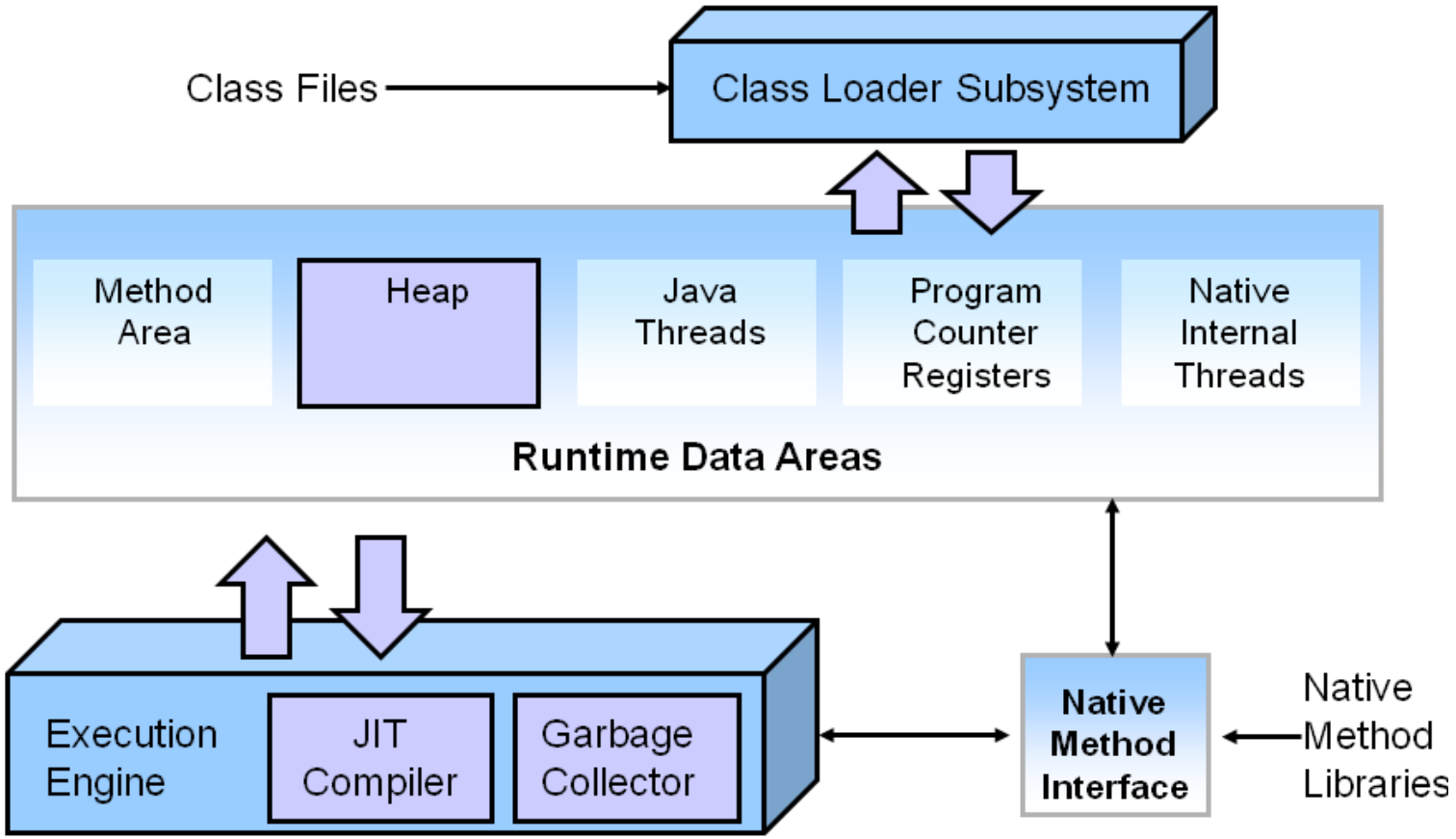
Memory management

Garbage Collection

JVM components



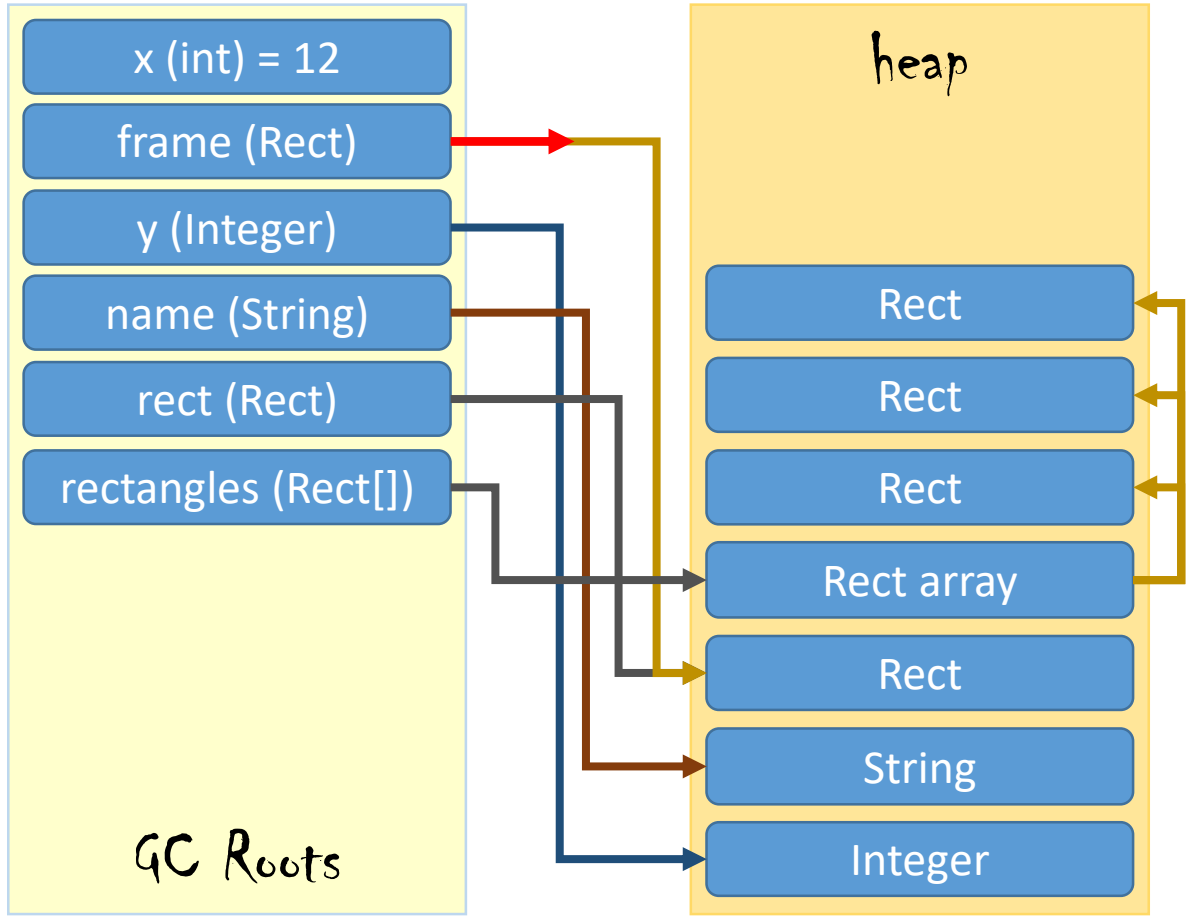
Instantiation and references

Allocated region could be referenced many times.

By `frame` and `rect`

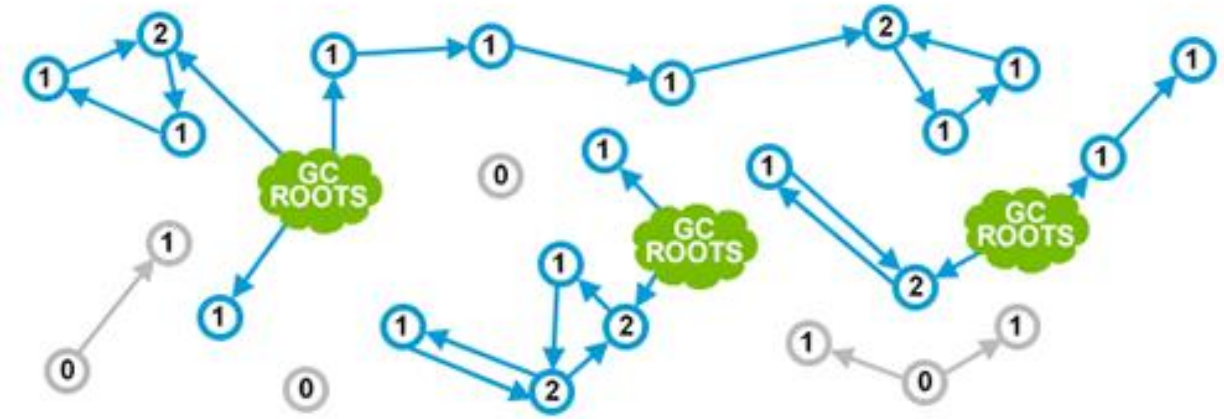
JVM counts references

Memory fragmentation



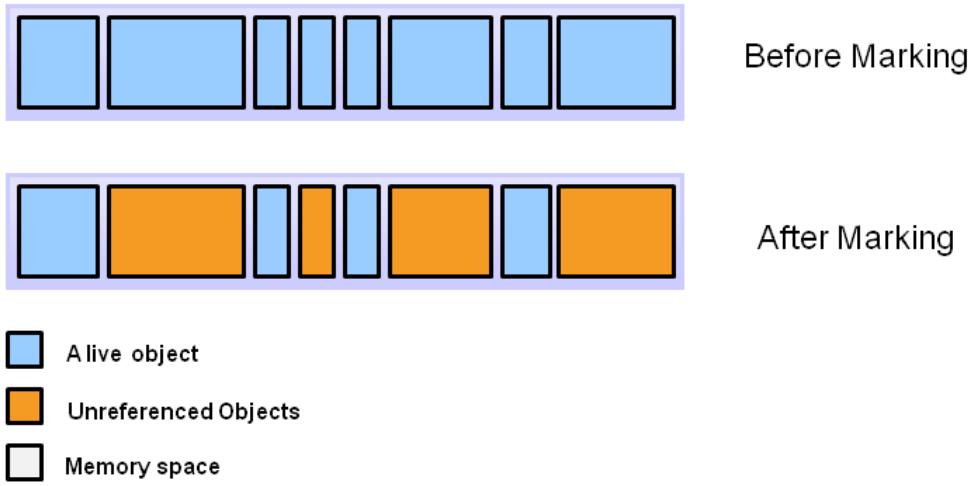
Reference counting

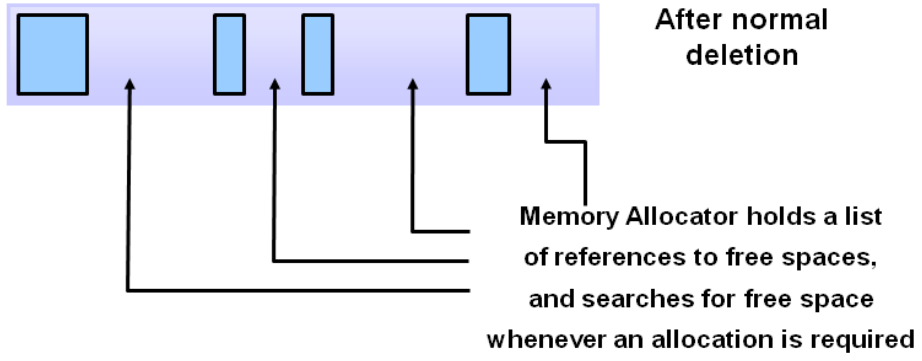
- References start from GC roots.
- Count references of objects.
- Also consider references of parents.



Marking for garbage collection

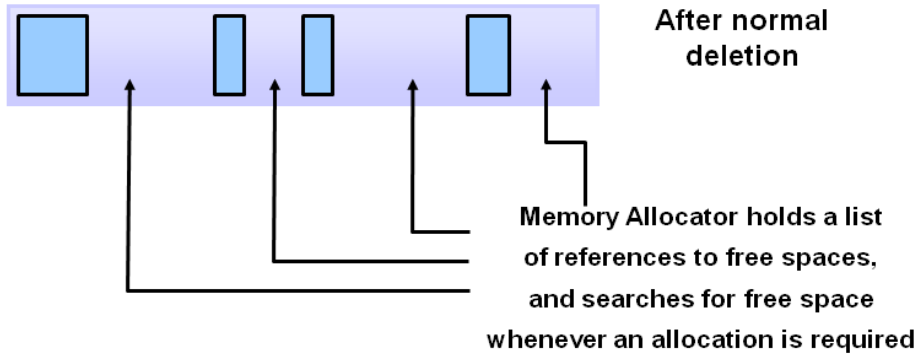
Based on counted references, mark objects as garbage

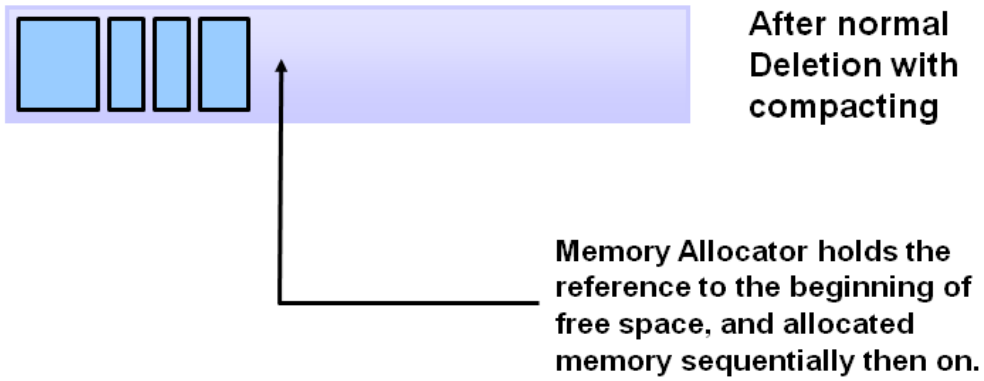




Result of defragmentation

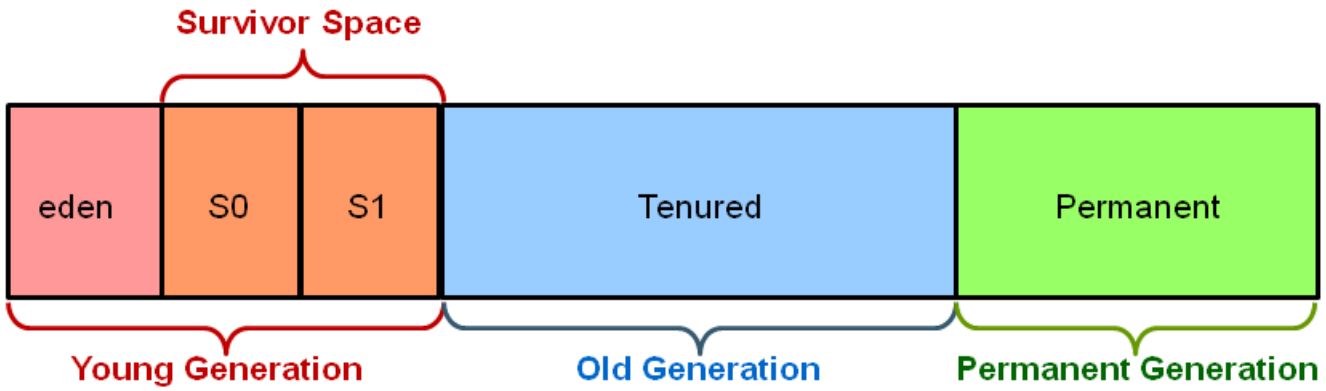
- **Logical** result of defragmentation
- Resources are taken from the applicati
- Many objects have to be moved
  - applied when program runs out of memory
  - when applied directly





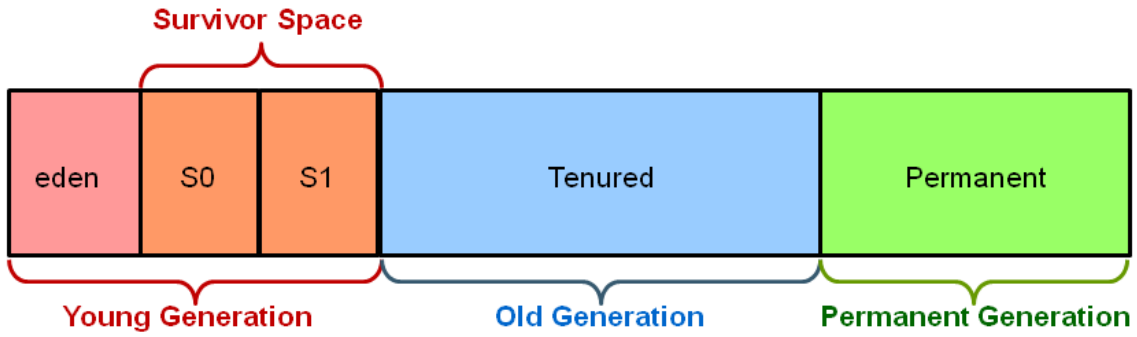
JVM object generations

- Garbage Collector runs frequently and well scheduled to provide consistent application performance
- Handles objects by age – object storage spaces
  - New/young
  - Survivor
  - Old
  - Permanent



JVM object generations

- Movement of objects is decreasing with their type (age)
  - **All objects** are moved to an older storage at a certain age
  - New/young – most frequently deleted or moved
  - Survivor – intermediate
  - Old – less frequently deleted or moved
  - Permanent – not moved



GC strategies

- G1 – quick defragmentation, release unused heap, optimized reservation
- Parallel – default in JDK8, parallel generational collection, no release
- ConcMarkSweep – concurrent mark & sweep
- Shenandoah – new best choice for vertical JVM scaling, immediate answer
- Custom GC also can be written

GC mode selection

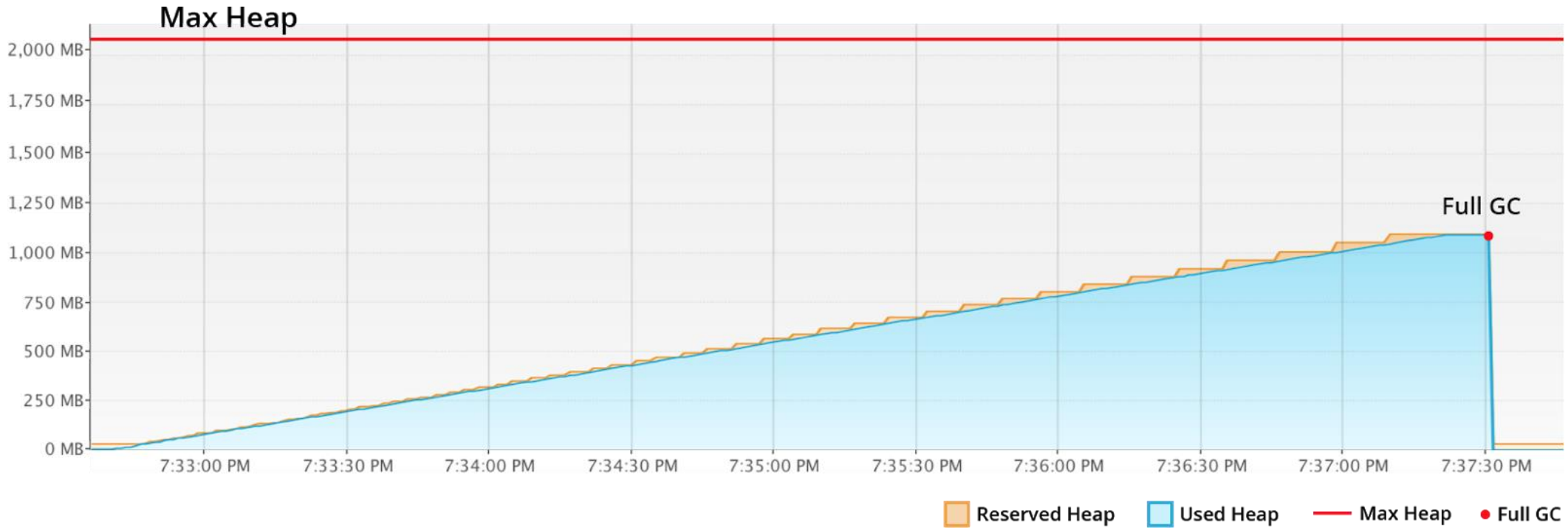
```
java -XX:+Use<gc_name>GC -Xmx2g -Xms32m -jar app.jar <sleep>
```

- Xmx – maximum scaling limit
- Xms – scaling step
- <sleep> – memory load cycle delay

https://dzone.com/articles/choosing-the-right-gc

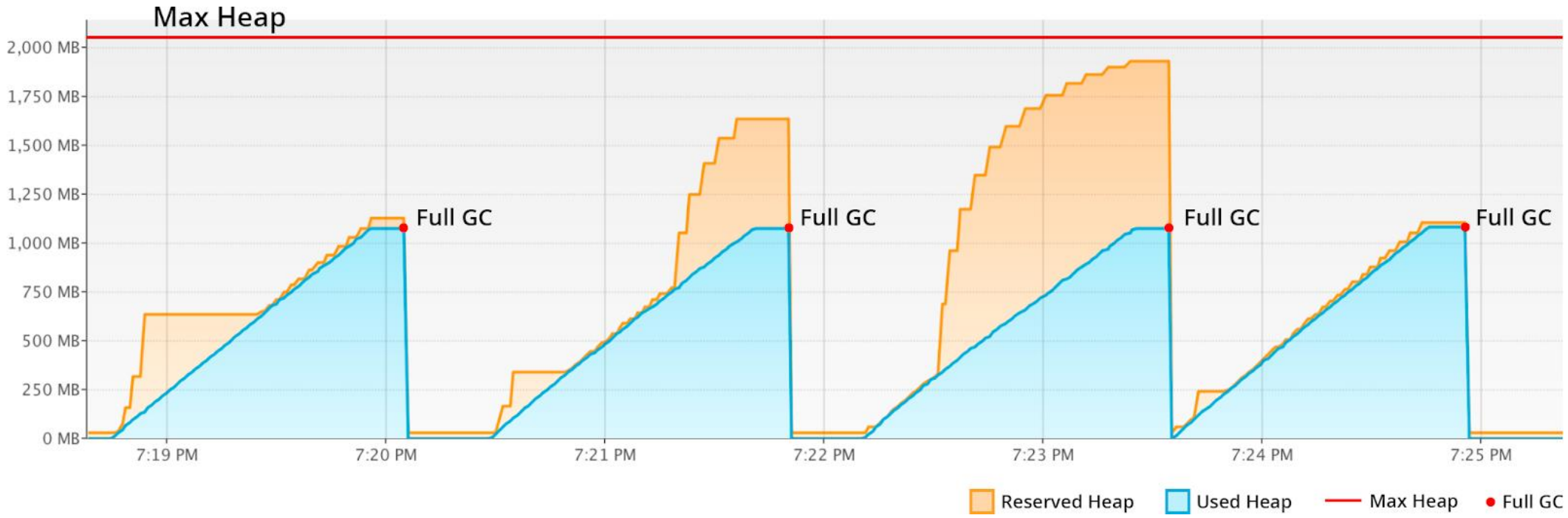
G1 – Memory reservation

Scaling based on **slow** memory usage growth



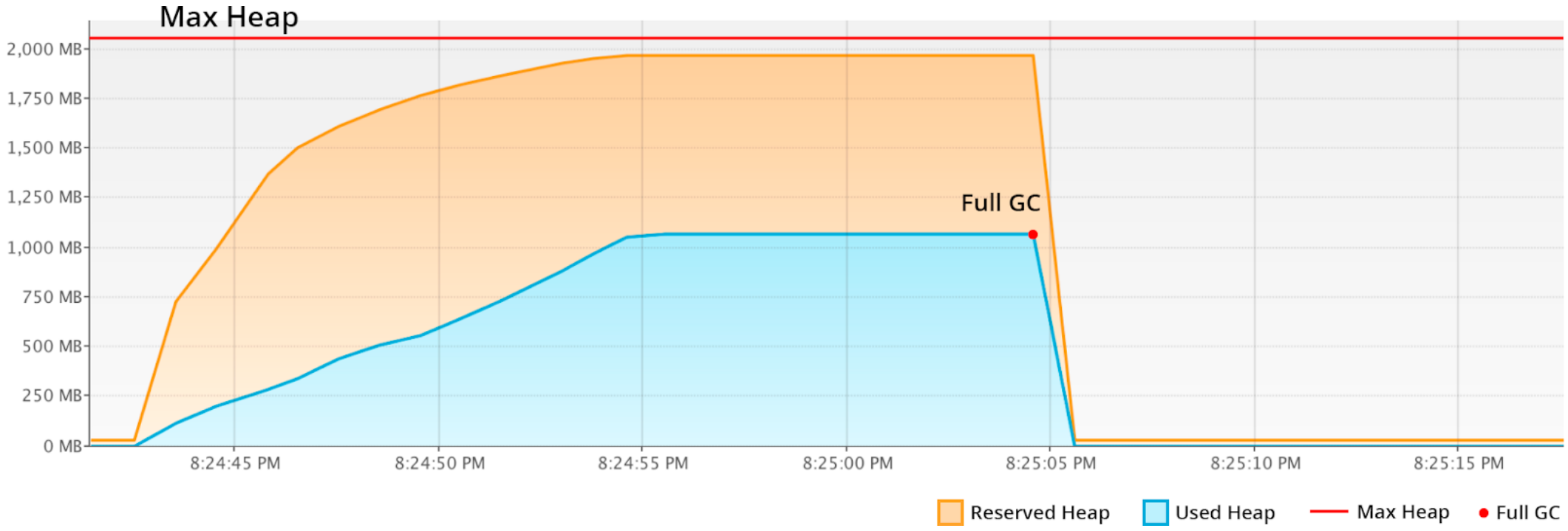
G1 – Memory reservation

Scaling based on **medium** memory usage growth



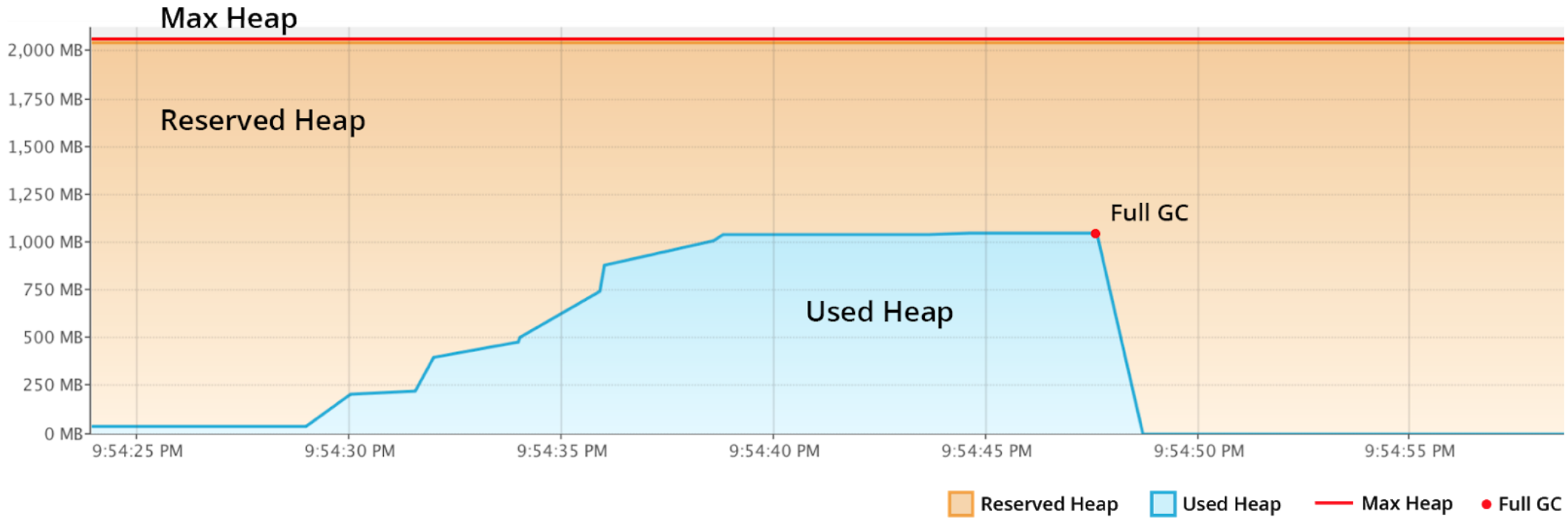
G1 – Memory reservation

Scaling based on **fast** memory usage growth



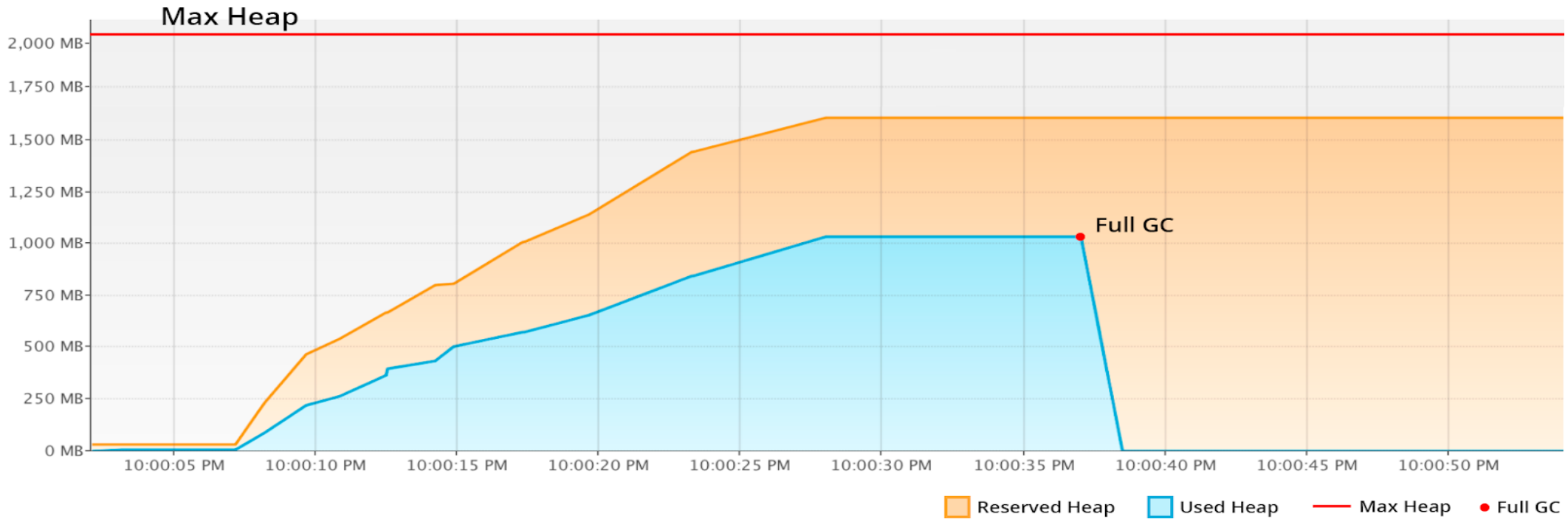
G1 – Memory reservation

Agressive heap reservation



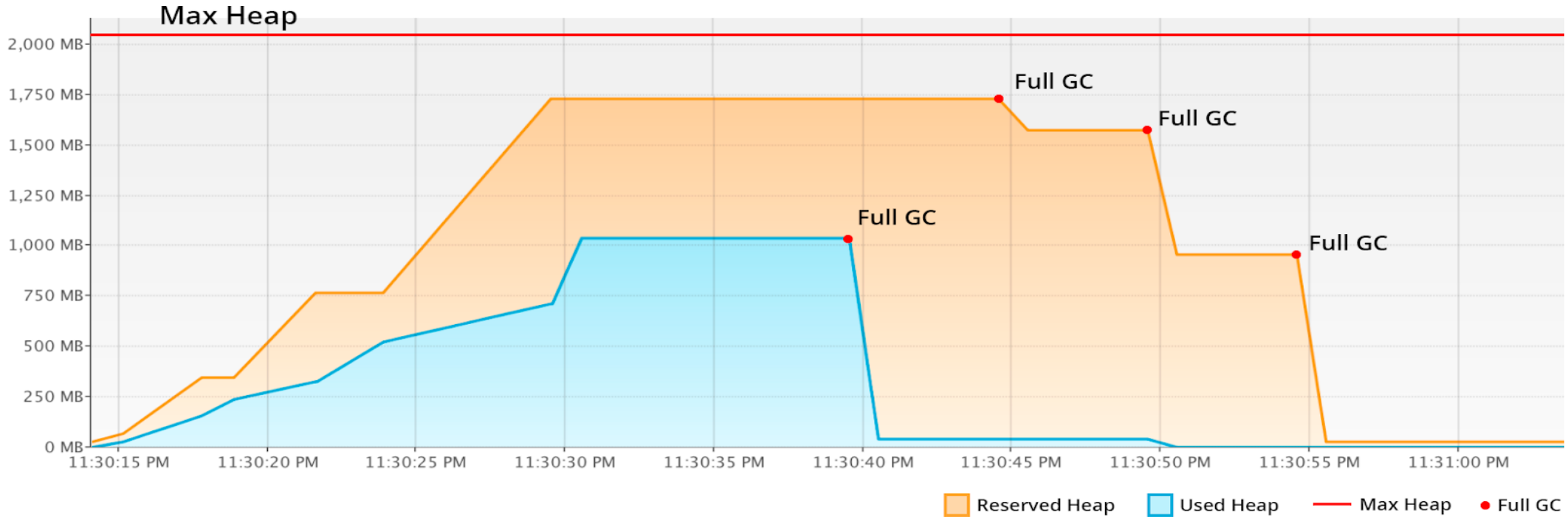
Parallel – Memory reservation

Does not support release of unused heap



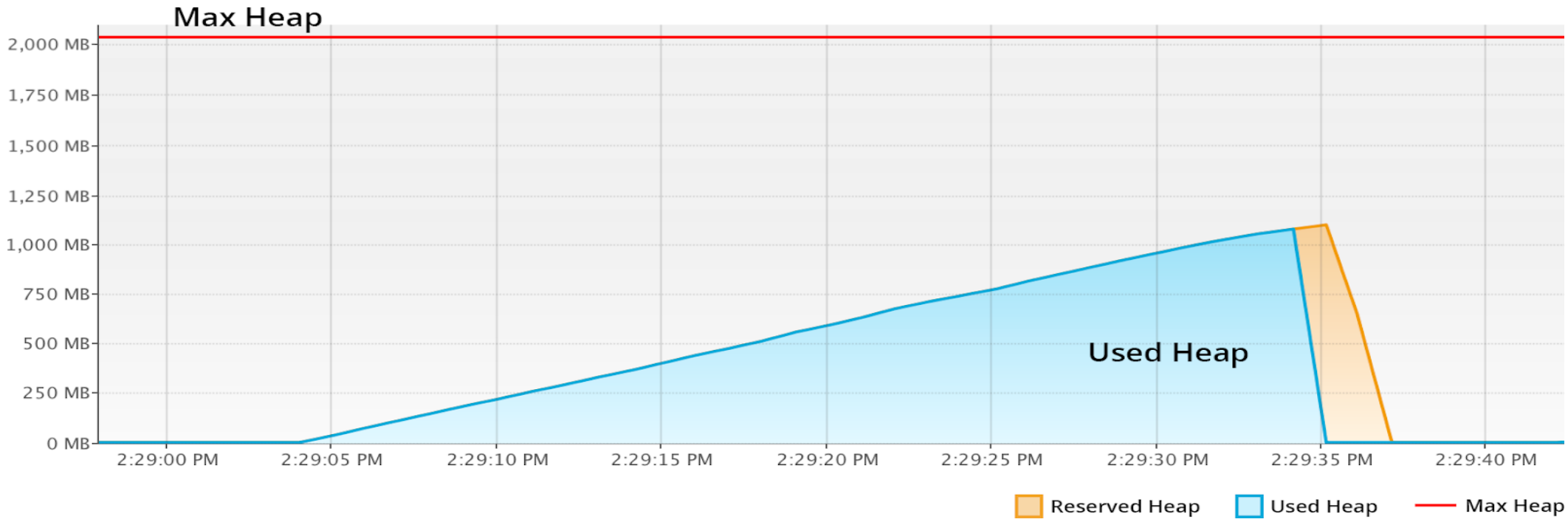
ConcMarkSweep – Memory reservation

Requires 4 cycles to release



Shenandoah – Memory reservation

Compact objects, clean and relese heap asynchronously without FullGC



Source Code Management

SCM – git

Fearful sentences of development

This function failed working. Did you **change anything**?

Can you send me the changes of **last Thursday**?

We worked hard in **last weeks**, let's put our works together!

I **can not find** your work. You have to rewrite!

This way it does not work, let's try **another way**!

This function has not been payed yet. **Remove** from production, but keep!

Development – the flow

- As a developer, main task is software development
- Circular method:
  - Take a task – understand, find integration point (where to change)
  - Solve – design and implement (how to change)
  - Test – find cases and data
  - Deliver

Waves and vortexes

- On local hardware failure finished or ongoing **works get lost**
- Developer has to **reimplement** the solution of a task
- **More developers** work on same project files
- A change has **side effects**
- Application has to be supported and developed at the **same time**

Waves and vortexes

- On local hardware failure finished or ongoing works get lost – **backup**
- Developer has to reimplement the solution of a task – **snapshot**
- More developers work on same project files – **manage concurrent access**
- A change has side effects – **wide dependency**
- Support and development at the same time – **non-linear development**

Source Code Management - SCM

A software solution to support management issues of source codes during development:

- To track and store revisions of a software (source codes)
  - Changes made in source code
  - Timestamp
  - Responsible person
- Revisions are available independently

Software solutions

- Microsoft Source Safe – centralised, discontinued in2017
- Apache Subversion – centralised version control system
- git – open source, distributed version control system

git

- By Linus Torvalds in 2005 for development of Linux kernel
- Server AND client in one
- A package to manage repositories
- Distributed repository, changes are synchronized
- Stores series of snapshots of modifications

Characteristics

- Strong support of non-linear development
- Distributed repository
- Compatible with existent protocols (FTP, HTTP, SSH, …)
- Efficient with large projects
- Cryptographic authentication is available
- Pluggable merge strategis
- Open source

Linear development

- One task at a time
- Make changes
- Save changes – **commit** changes
- Does not depend on task type (development, bugfix, support change)

Staging

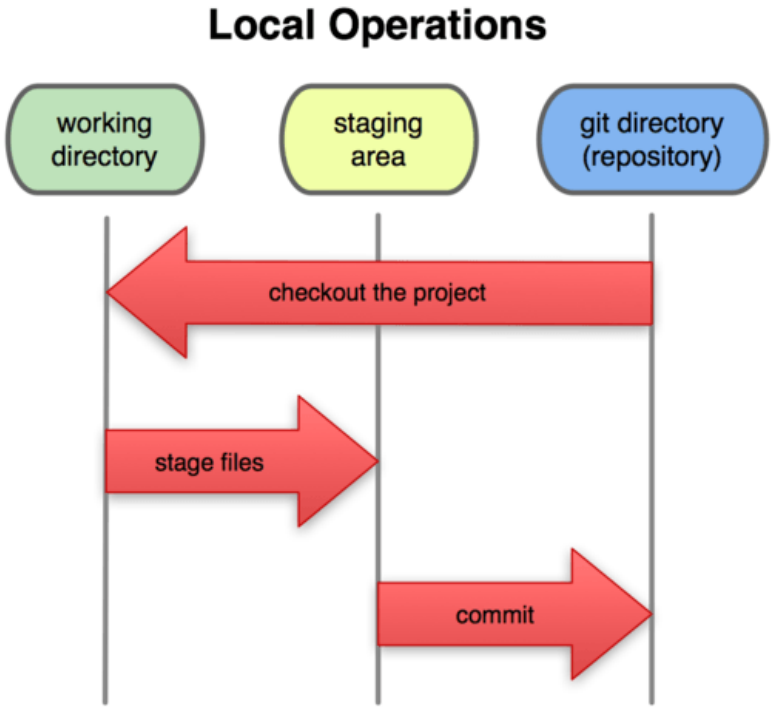
- Staging area is a simple file in git directory
- Stores contents of the NEXT commit

Commits contain changes made on source code, but **not all changes** have to be commited.

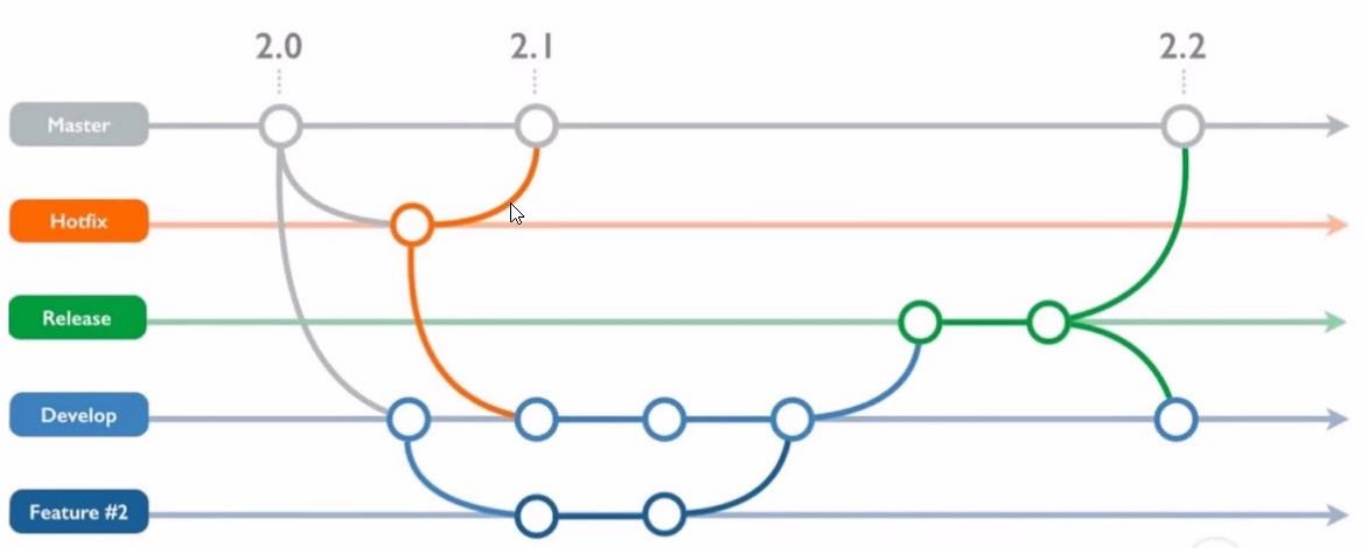
Before commit, changes are staged: **snapshots** are made **of changes** wanted to commit

Commits

- Store staged snapshots in the local git repository
- A message can be joined as a description
- User and timestamp are always stored
- Use `--amend` to modify last commit



Non-linear development

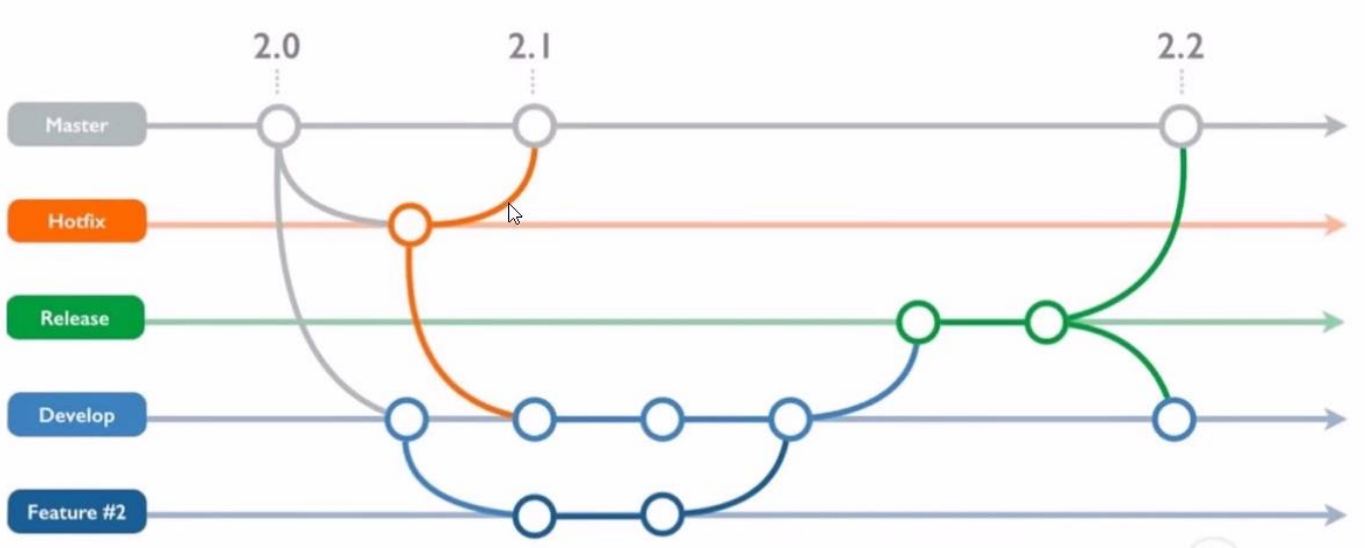


Branches

- Git stores changes in commit objects with reference to changes
- Branch is a movable pointer to one of these commit objects
- Points to the last commit made related to the current branch
- Creating new branch is simply creation of a new pointer
- Default: `master`

Manage branches

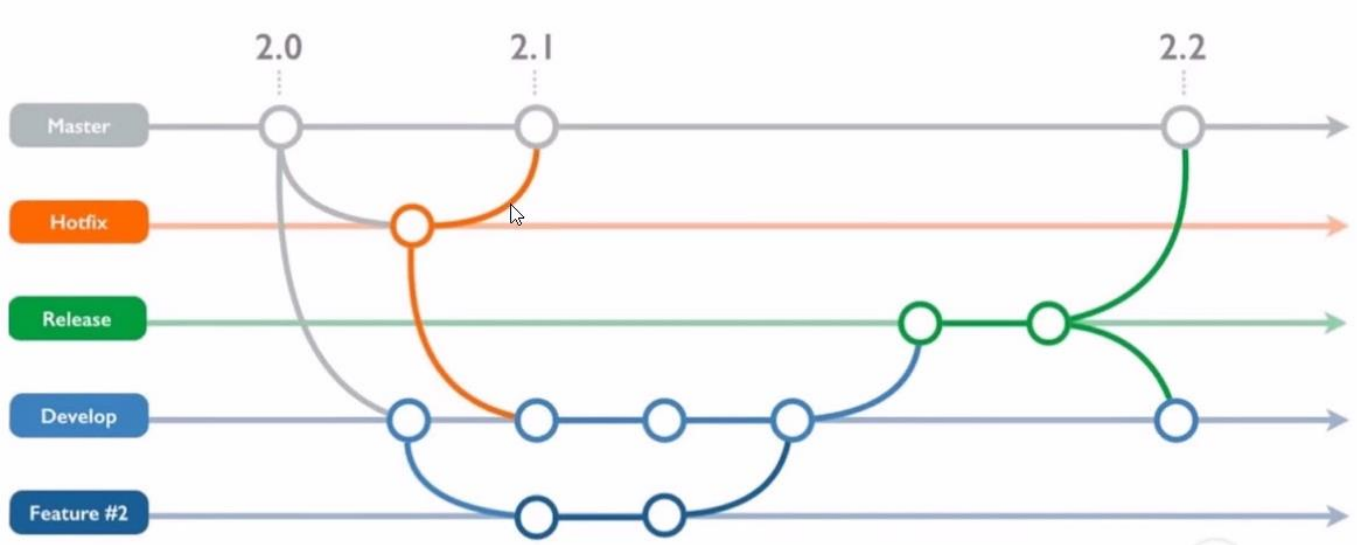
- Create new: `git branch <branch name>`
- Switch to a branch: `git checkout <branch name>`
- Current branch: `HEAD`



Merges

- Incorporate changes of commits into the current branch
- Replays changes made in named commits

Conflict: generally arise when two developers have changed the same lines in a file.



Git conflicts

- On merge start: merge conflicts with pending local changes
  The local state will need to be stabilized using
- During merge: conflict between the current local branch and the branch being merged
  Git will do its best to merge the files but will leave things for you to resolve manually in the conflicted files

Remotes

- A remote **copy** of the local repository
- Connected via network (not local repository)
- Could also contain branches
- **Local branches can track remote brances**
- Transfer commits back and forth
- Benefits:
  - Backup storage
  - Place for managing contributions (concurrent access)
  - Place to publish

Remote commands

- clone – create a local copy of a remote repository
- fetch – update remote tracking branch
- pull – pull changes from remote repository and merge with local changes
- push – push commits into remote repository

Other git commands

- reset – discards changes in a local branch
- revert – reverts changes in public branch by creating a new commit
- rebase – reapply commits on new base (move)
- cherry-pick – reapply commits on new base (copy)
- tag – manage a tag object signed with GPG, message could be attached

GitLab – https://gitlab.com/

- A git server to store git projects in the cloud
- Supports administration
- Free, but with payed plans supports agile development and statistics
- Supports continous integration and delivery

Projects

- Details and statistics
- Activity – a log about activities executed in the project
- Repository – the source code git repository
- Issues – issues and tasks to deal with in the project
- Merge requests – when finishing a development step, request for merging
- Wiki – s structured storage of project info

Issues

- Tasks to solve, with description and tags
- Can have responsible developer
- Management board with due dates
- Can be related to branches and merge revisions

Merge request

- Motivation – exchange changes with collaborators
- Meaning – request to merge a branch into another
- Task – check the modifications to satisfy principals and conventions
- Responsibility – developer who made changes is responsible for them
- Properties – template, responsibility

Merge revision

According to MR settings

- Merge a branch into another
- Modifications are checked by other developer
- Code check – to satisfy programming conventions and coding conventions
- Functional check – run tests
- Discussions have to be closed to merge

Merge discussions

- Open a discussion about a diversion from principal or convention
- More discussions could be opened
- Code changes are made by owner (responsible)
- Or answer in a discussion why and what was implemented
- Discussion could be closed by any parties
- Reviewer merges when all discussions are solved

Developer workshop

Agile development and scrum

Agile development principals I

- Customer satisfaction by early and continuous delivery of valuable software.
- Welcome changing requirements, even in late development.
- Deliver working software frequently (weeks rather than months)
- Close, daily cooperation between business people and developers
- Projects are built around motivated individuals, who should be trusted
- Face-to-face conversation is the best form of communication (co-location)

Agile development principals II

- Working software is the primary measure of progress
- Sustainable development, able to maintain a constant pace
- Continuous attention to technical excellence and good design
- Simplicity—the art of maximizing the amount of work not done—is essential
- Best architectures, requirements, and designs emerge from self-organizing teams
- Regularly, the team reflects on how to become more effective, and adjusts accordingly

Agile properties

- Iterative and incremental – development in small increments (sprints)
- Short feedback – daily stand-up about works made and new plans
- Focus on quality – continous integration and delivery

Sprint

- A short time frame in which a useable and releasabe product increment is created.
- Less than a month (2-4 weeks)
- Consistent duration
- Starts immediately after previous
- **Scope can be changed, but goals do not**

Sprint kickoff

- Check preliminary project schedule
- Discuss tasks: benefits, complexity, risk, priority
- Sort BackLog
- Set sprint scope
- Set print goals (1-3)

Scrum Poker

Motivation:

- estimate task complexity
- avoid influence of other participants

Iterative flow:

- discuss task contents
- estimate complexity
- estimations on boundaries continue content discussions

Scrum

Discuss:

- Works made since previous scrum
- Plans until next scrum
- Required resources and their availability
- Planned implementations

- Lasts for approximately 15 minutes

Sprint retrospective

- Organized after closing a sprint but before starting a new one
- Inspect the team work and efficiency
- Create plan to improve next sprint
- What went well
- What to improve
- What to commit to improve next sprint (action)

Burndown

- Measurement of developer performance
- A graphical representation of work left
- Prediction of finish and resource requirements
- X angle: time
- Y angle: story points – work to be done