Taking aim at ENOSPC: New layouts for an old problem

Self-Extending Progressive File Layouts (SEPFL)

Progressive File Layouts

- Allow different layouts for different areas of a file
- Rich ground for new features:
  Data on MDT
  FLR: Mirroring, erasure coded, etc.
- Major improvements to ease of use, performance, reliability, etc.

Pools & Tiering

- Pools & tiering are an obvious application of PFLDifferent regions of file on different storage classes (pools)
- By default all files use same layout, no recourse if a pool runs out of space
- Seems odd but a major concern for some customers

What about ENOSPC?

- Traditional Lustre response – Avoid it through improved OST allocation
- PFL can help with this by distributing different file regions to different OSTs (basically, smaller allocation "granularity")
- But it's all static for a given file – If a file is on a set of OSTs and they run out of space, allocation policies can't help.

Dynamic Layouts

- Dynamic layouts: Layouts that change automatically based on system state
- Possibilities abound, a lot of them fall foul of "dirty data" problem
- Can't change layout component that client is currently writing to, no mechanism to handle that i/o

Loophole 1: Uninitialized Components

- But you can change the extent of components which are not initialized.
- Uninitialized “virtual” components which are never init
- When i/o hits one of these components, client sends request for layout to the server
- Instead of initing the component, the server replaces the needed extent with normal layout space

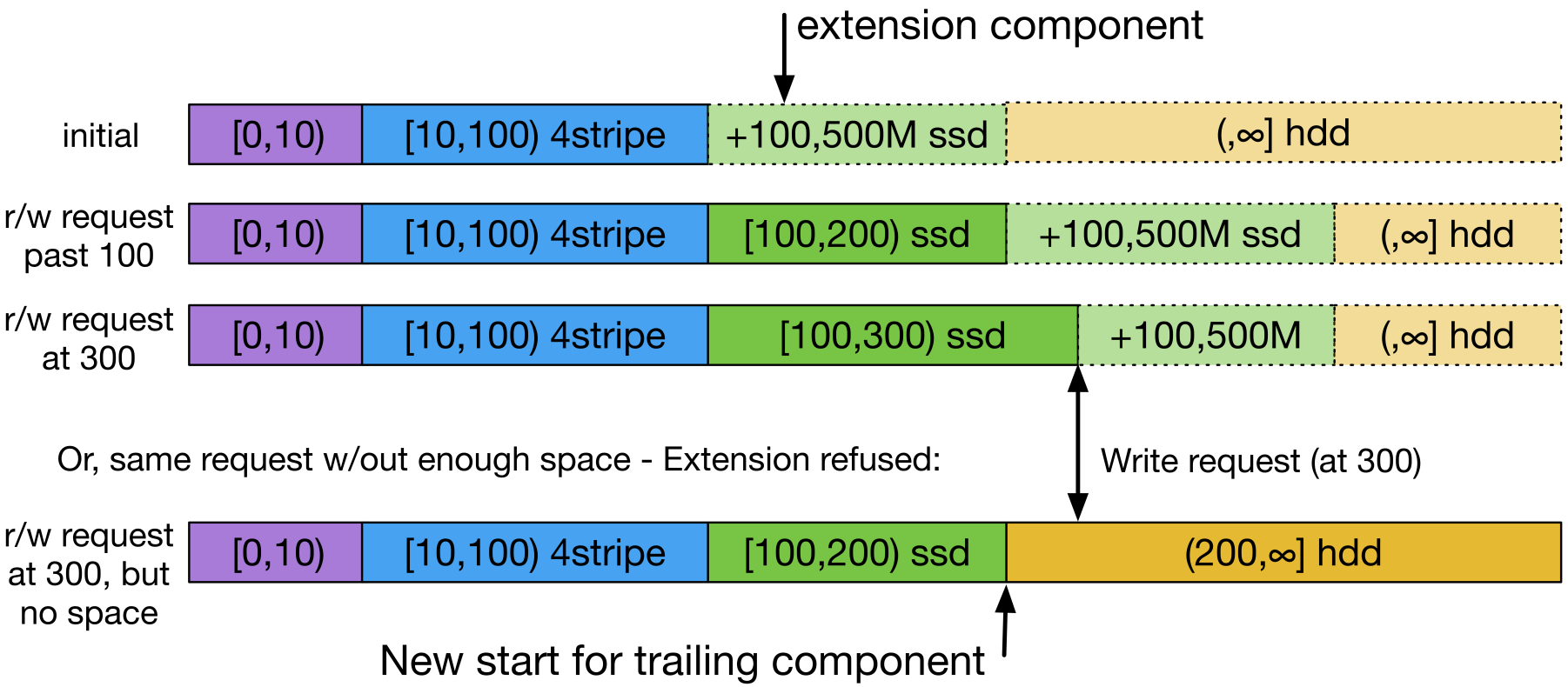
Loophole 2: Increasing Extents

- It is never safe to reduce the extent of an initialized component
- But you can increase it by moving the start "up" or the end "down"
- When I/O hits a "virtual" component, change the extent of neighboring "real" components to allow the i/o to complete

Self-Extending Layouts

- Self-extending PFL (LU-10070)
- Some PFL segments are virtual, never instantiated
- Request in a virtual segment requires layout update
- MDS grants new layout (extend existing component, spillover to trailing, or create new)
- Can make choice based on dynamic conditions (e.g. free space)

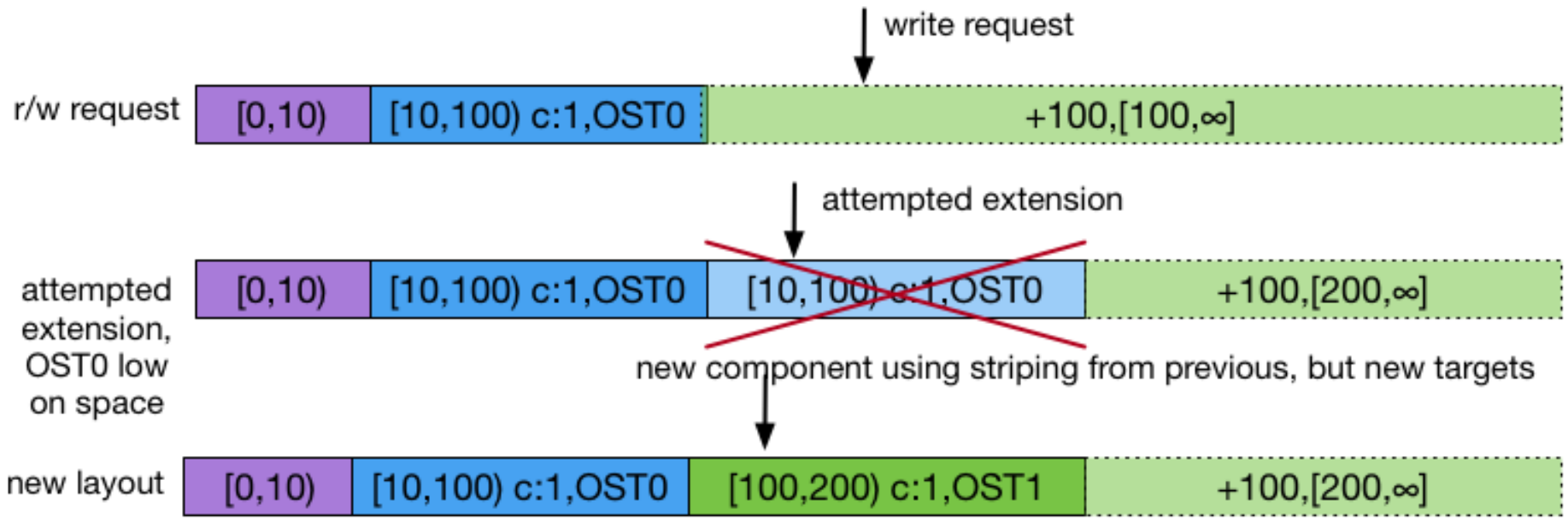
Self-Extending Layouts



More than Tiering

- Tiering is the obvious application, described previously
- But there's a good trick for files without tiering
- "Self-spillover" or "spillover restriping"
- When the existing OSTs run low on space, create a new component with same striping properties

Spillover Restriping



Why not just a better allocator?

- Partly orthogonal to improved allocation – It's still possible to run out of space
- Pools of small OSTs need spillover
- Also gives the allocator more chances to split files, gives more chances to improve allocation in a badly balanced file system

What about mirror + resync?

- Interoperates transparently with mirrors + resync, etc.
- Any layout can have self-extending components
- Extension policy is just applied before instantiating any components
- Works the same for mirrors as regular components

Caveat Emptor

- Not perfect, possibly a stop-gap solution
- If we could restart i/o on a layout change (very tricky), we could be truly responsive:
  Only change layout on actual ENOSPC
- Group locks don't work today (requirement to fully instantiate layout)
- Append has a similar problem, but should be fixable:
  LU-9341

Finally:

- **Any questions?**
- Happy to answer questions later or by email (paf@cray.com)