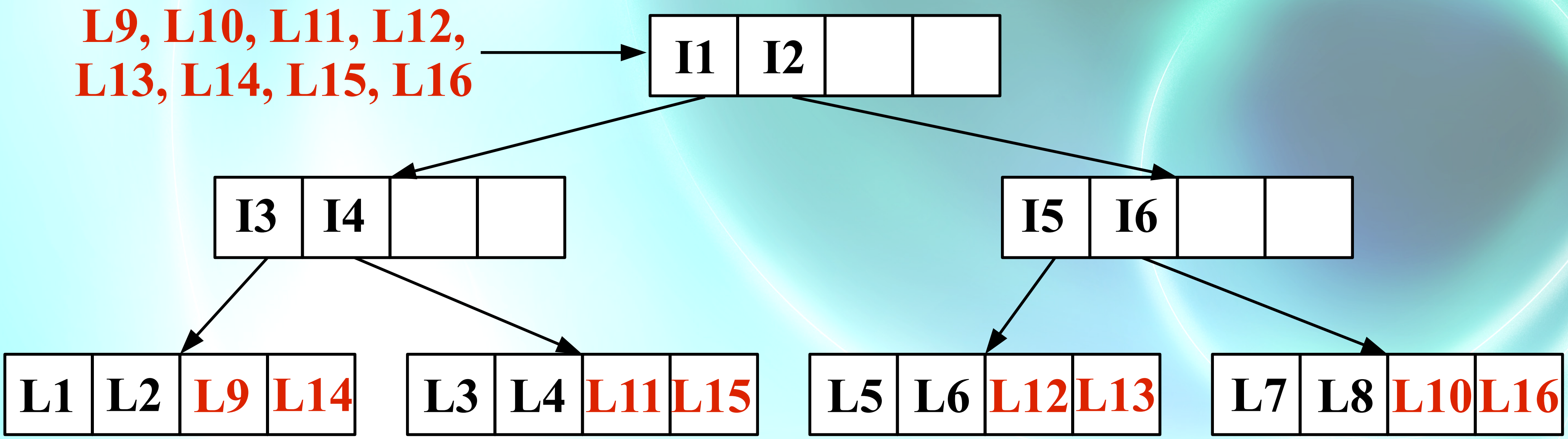
Fast GiST index build

Alexander Korotkov
PostgreSQL Conference Europe 2011, Amsterdam

Ordinal GiST index build

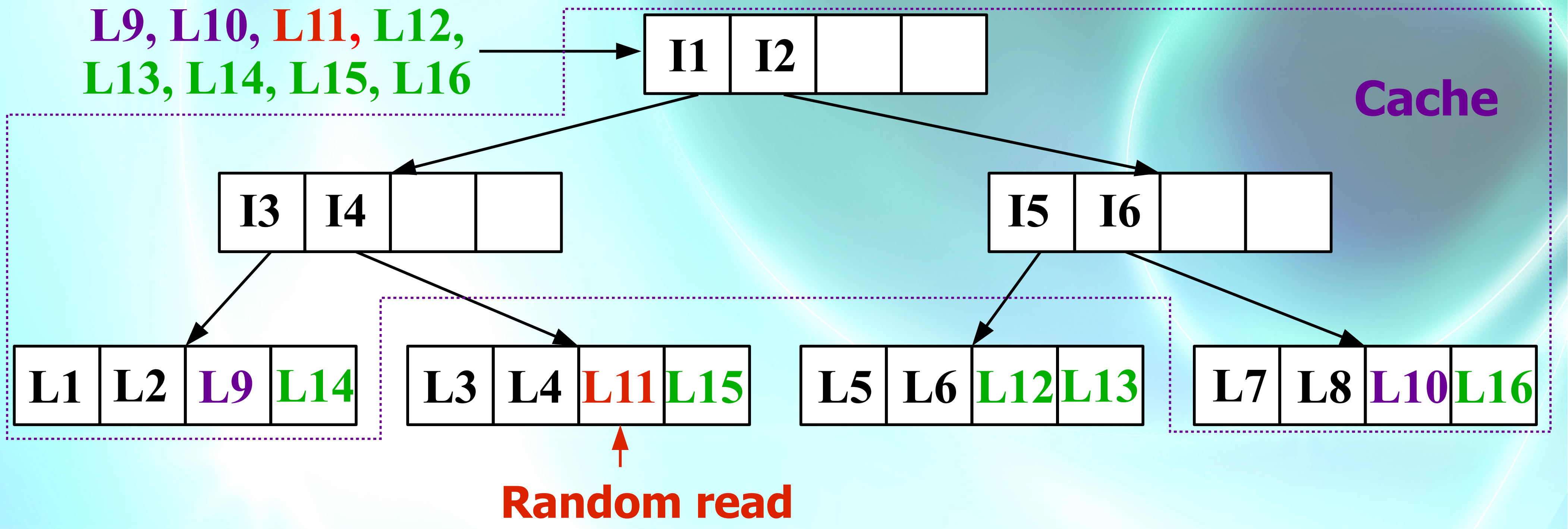
We have to insert index tuples one by one



When something go wrong with ordinal GiST index build algorithm?

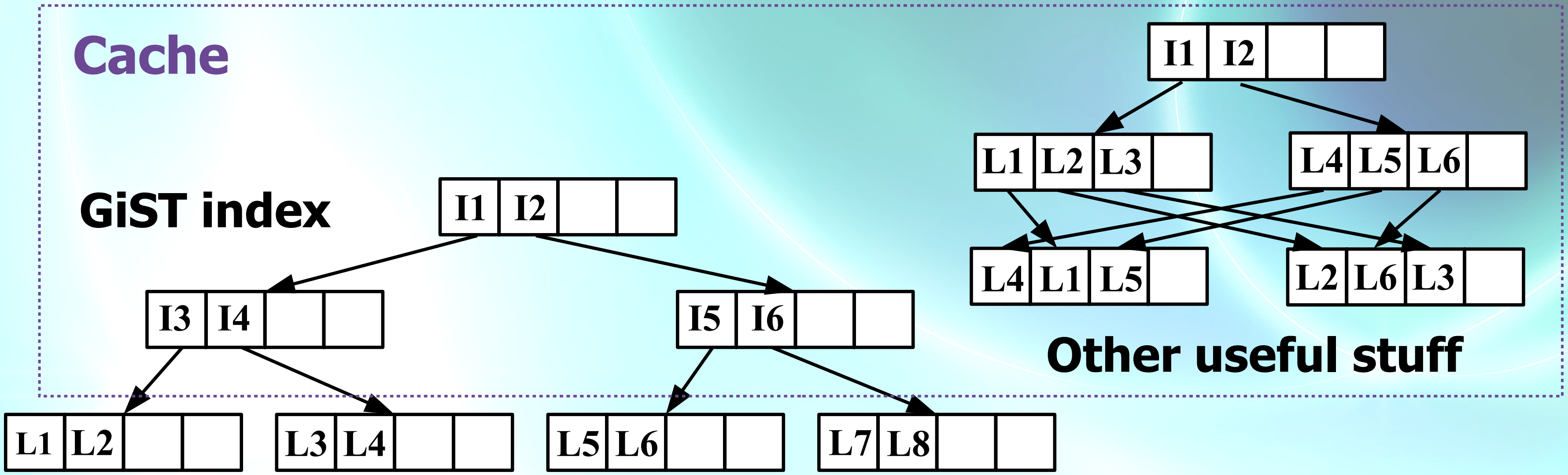
Index too large to fit into cache

Significant fraction of index tuple inserts cause random IO



High concurrency

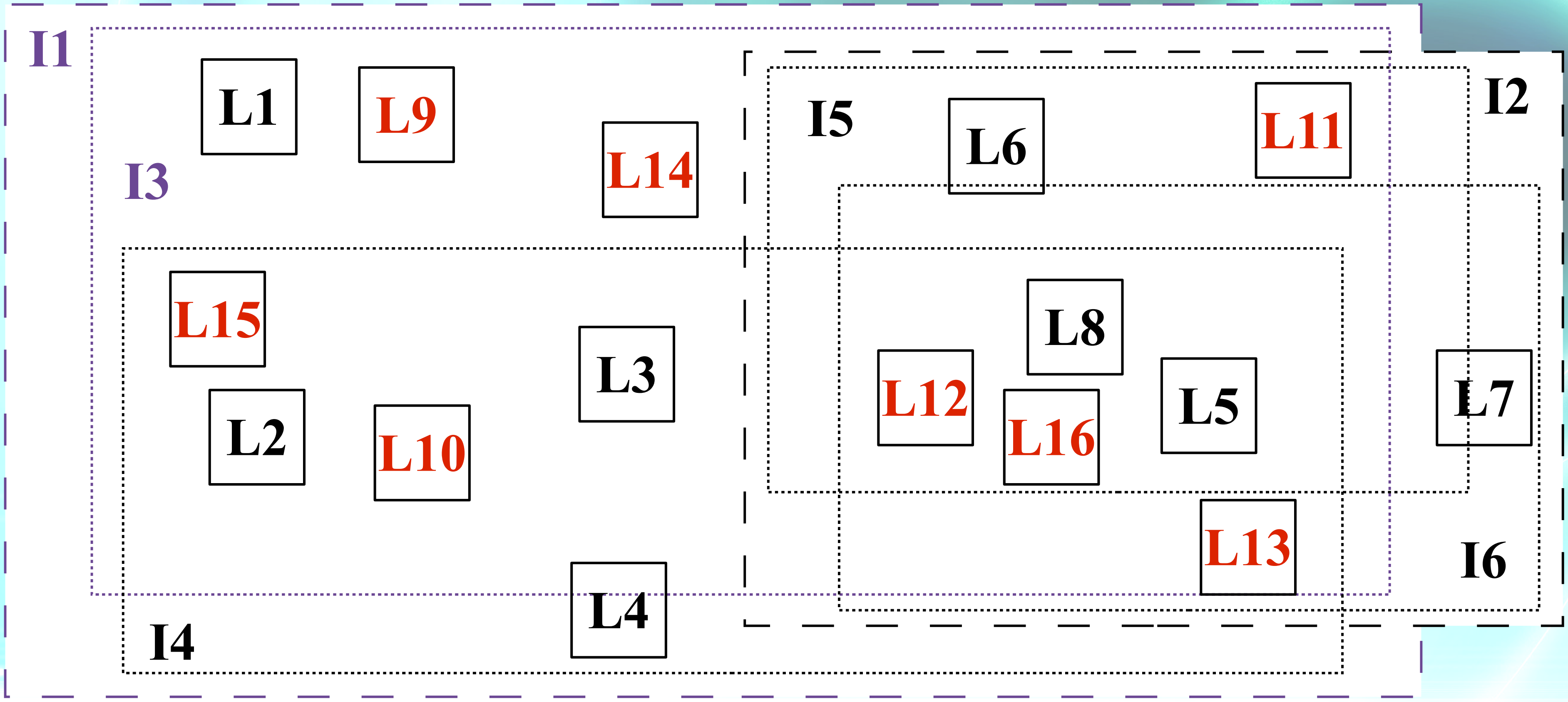
If even index can fit to entire cache, other backends also use cache. So, even not too large index can be out of cache.



What helps to orginal GiST index build algorithm?

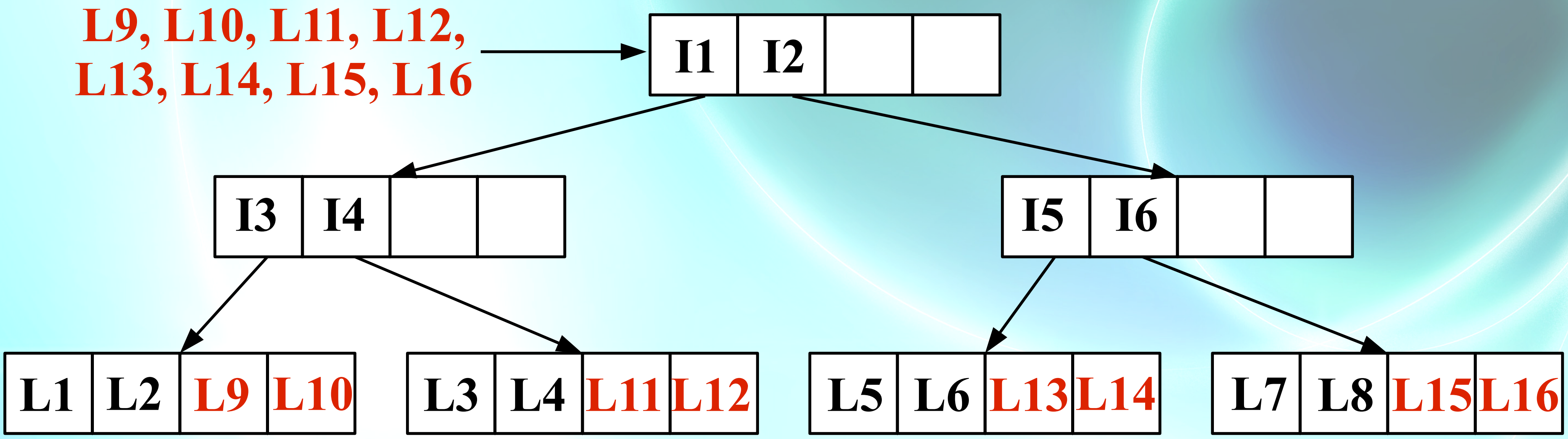
High overlap

gistchoose selects the first path with zero penalty (if any).



Ordered datasets

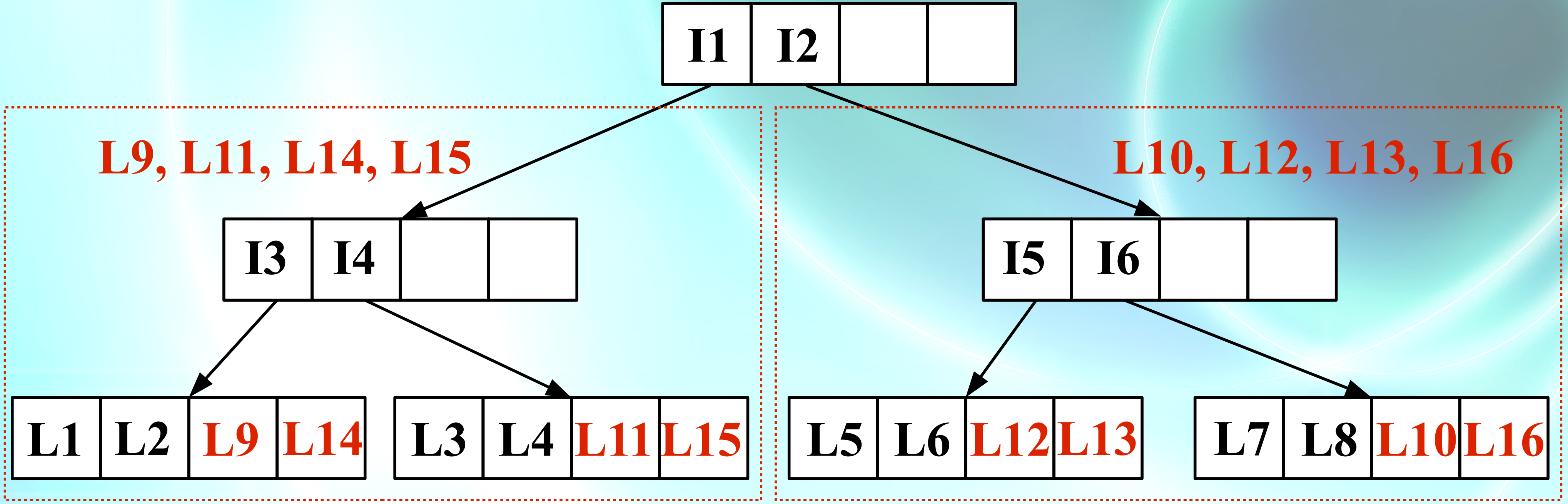
Ordered dataset cause inserts to be in the recently accessed part of index. That's very good for caching.



The buffering GIST index build technique

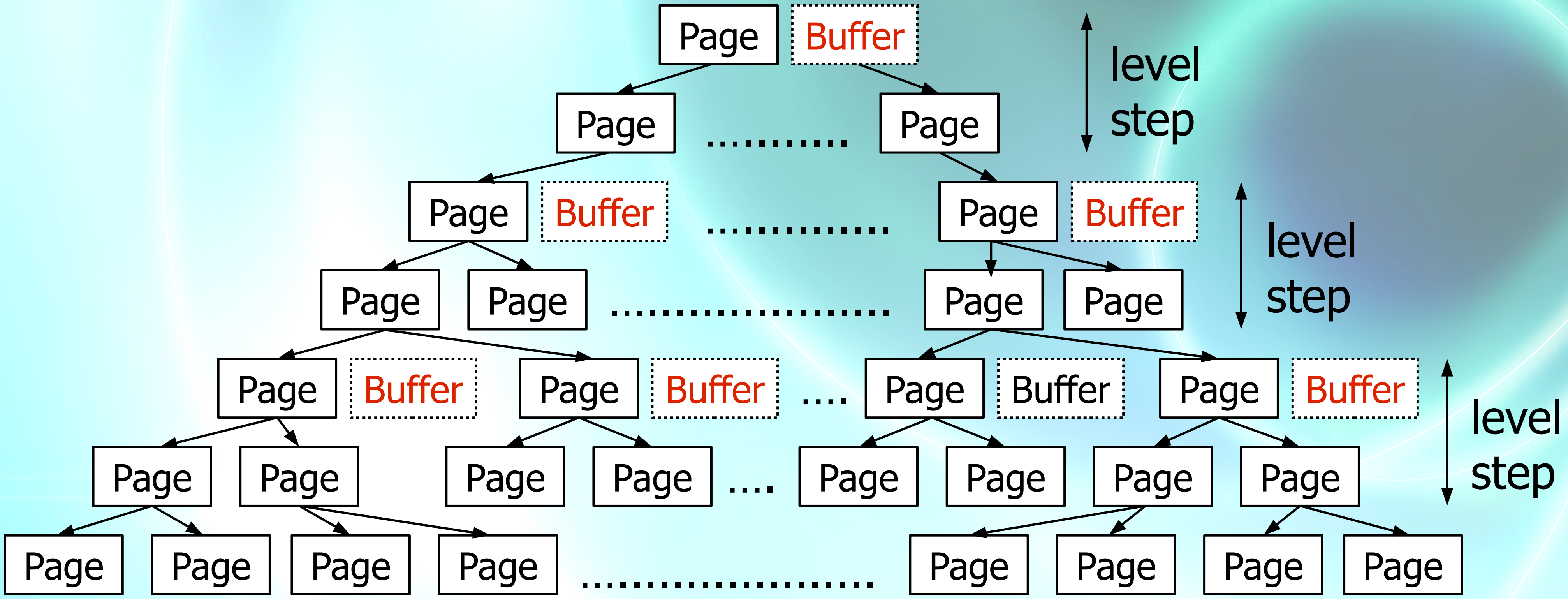
General idea

Remember index tuples which are going to specific part of tree, and then process pack of them later.



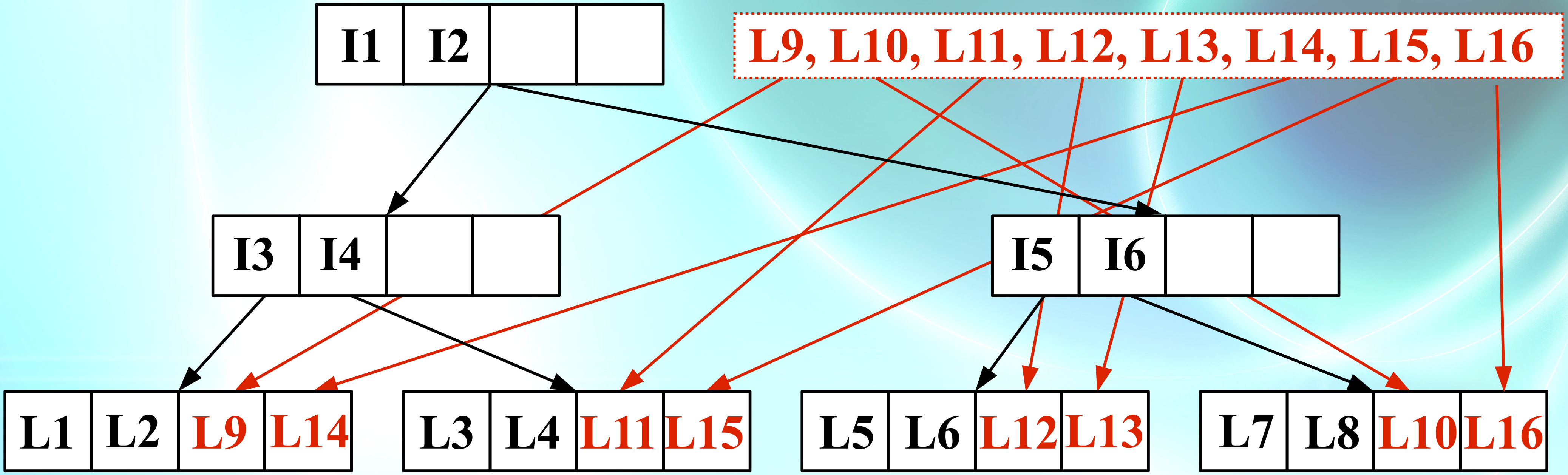
Buffering algorithm

Buffering algorithm is a recursive application of that idea.



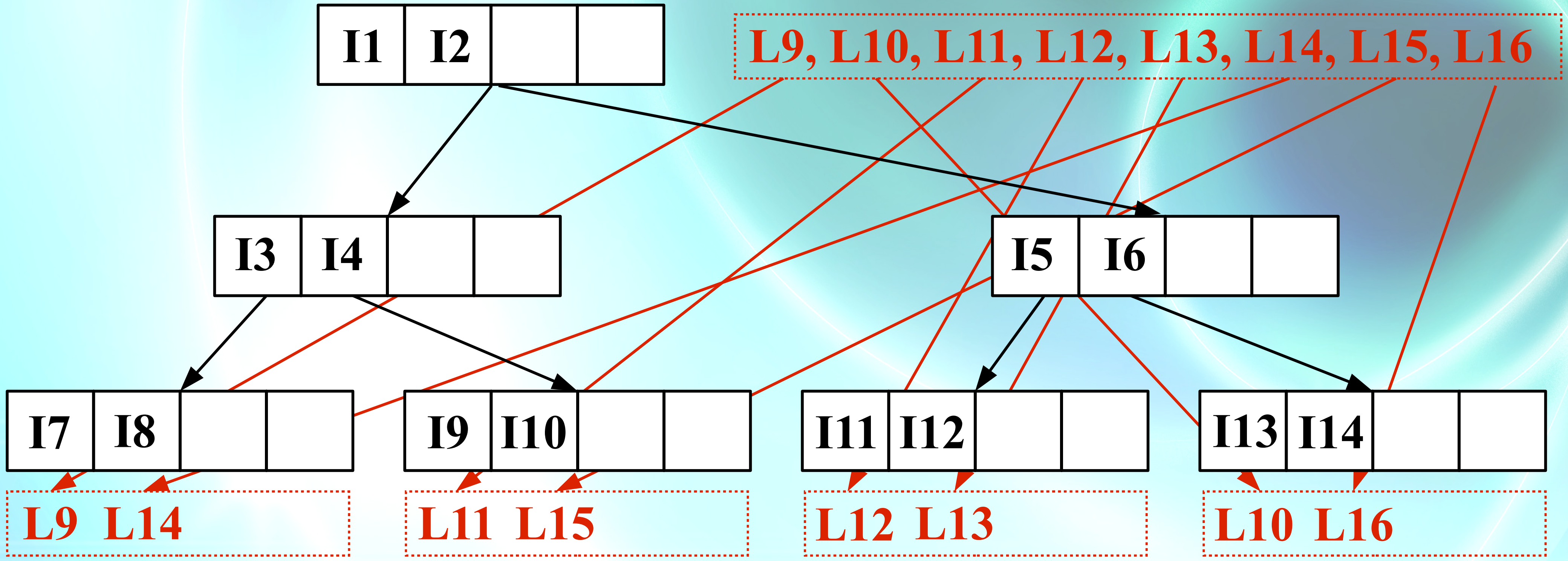
Buffer emptying

Lowest level overflowed buffer can be emptied to leaf pages.



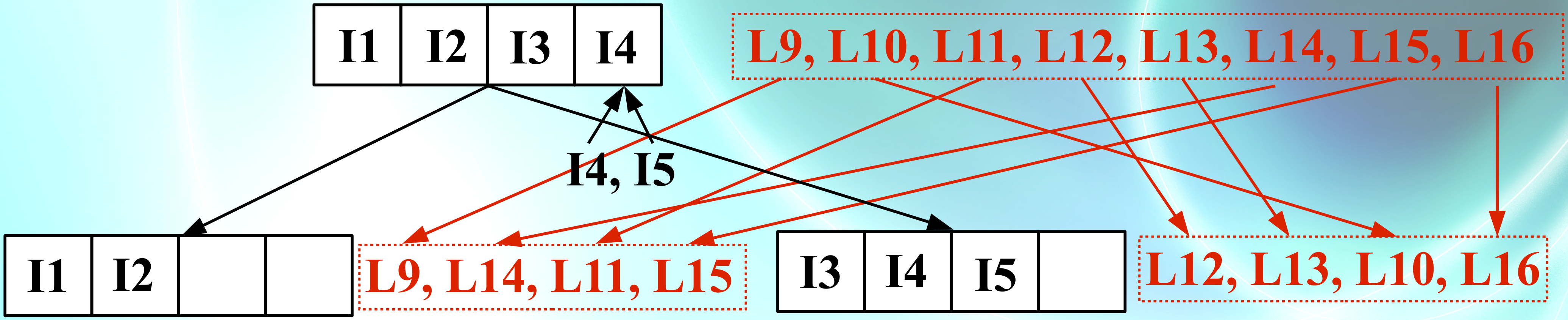
Buffer emptying

Higher level overflowed buffers can be emptied to lower level buffers.



Page splitting

When page is splitting, attached buffer is splitting too.



Final buffers emptying

When all the tuples are inserted, final buffers emptying stage is starting. All even non-overflowed buffers are emptying in up-to-down manner.

Buffer size and level step selection

- Subtree of level step height should fits to cache. Therefore, operations inside subtree are IO efficient.
- Buffer size should be comparable with size of subtree. Thus, IO would be comparable with size of inserted data.

Varlena data

Level step and buffer size parameters are depending on index tuple size.

- Level step is determined for worst case. Subtree should fits to cache, even if all varlena tuples are of minimal size.
- Buffer size is in runtime tuning based on average size of inserted index tuples.

Analysis

When does buffering help?

- Somebody may be dissapointed, but buffering helps only if bottleneck of index build is IO (it could be rather dramatic help).
- GiST is also CPU expensive in comparison with, for example, Btree, because many penalty and consistent calls (it have to do that calls for each index tuple in page which is in use). Buffering doesn't do any help with that.

Node splitting algorithm tradeoff

- With buffering build we can now be sure that even large index tree with low overlaps doesn't require enormous time to construct.
- With buffering build, it's a good time to descrease overlaps by new node splitting algorithm.

Double sorting node splitting

- Is based on more comprehensive consideration of splits along axis.
- Has complexity O(n*log(n)), n — tuples count.
- On large datasets it shows much better index quality, because of less overlap (page accesses can be less in times or even dozens of times!)

IO vs CPU tradeoff

- Buffering technique is optimizing IO, but it costs additional CPU load.
- When index fits to cache, buffering is just waste of CPU.

IO vs CPU tradeoff

"buffering" parameter of GiST index

- on — try to use buffering anyway (if have enough of RAM)
- off — don't use buffering anyway
- auto (default) — try to switch to buffering when index size exceeds effective_cache_size

Testing

Test setup with 2Gb of RAM builds index on 100M of row.

Dataset	Split method	Build method	Actual time	Search time
Uniform	New linear	regular	17 h 39 m	1
		buffering	3 h 23 m	0.90
	Double sorting	regular	9 d 10 h	0.089
		buffering	4 h 11 m	0.089
USNOA2 (ordered)	New linear	regular	56 m	1
		buffering	1 h 27 m	0.46
	Double sorting	regular	45 m	0.37
		buffering	1 h 29 m	0.36
USNOA2 (shuffled)	New linear	regular	10 h 12 m	1
		buffering	3 h 16 m	0.89
	Double sorting	regular	8 d 20 h	0.072
		buffering	4 h 20 m	0.067

Analysis

Buffering techinique:

- Accelerate build of non-overlapping trees on shuffed data in dozens of times!
- Might accelerate build of even high-overlapping trees (depending of OS cache strategy)
- Adds some slowdown to index build on well-ordered datasets.

Future work

- Improve automatic switching to buffering build mode (detect concurrent load and ordered datasets)
- Decrease CPU usage
- Extend GiST interface with special features for index build (for example, Hilbert's curve).

Acknowledgement

- Oleg and Teodor for giving direction and advices
- Heikki Linnakages for mentoring and his work on this patch
- GSoC for funding of this project

Thank you for attention!
Questions are welcome!