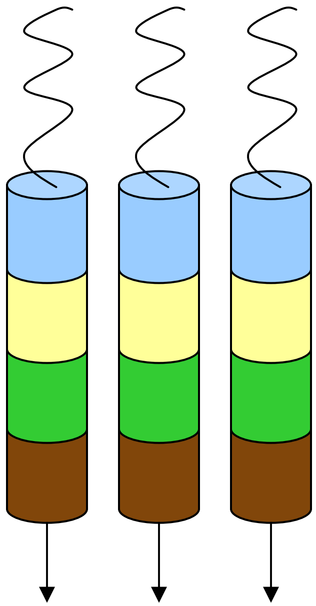
Threads vs. Events

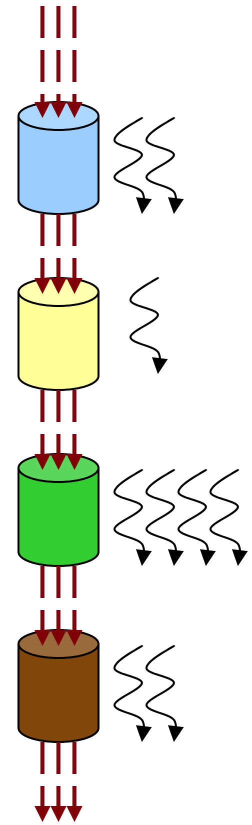
Capriccio – A Thread Model

Two approaches



- **Capriccio**
  - Each service request bound to an independent thread
  - Each thread executes all stages of the computation

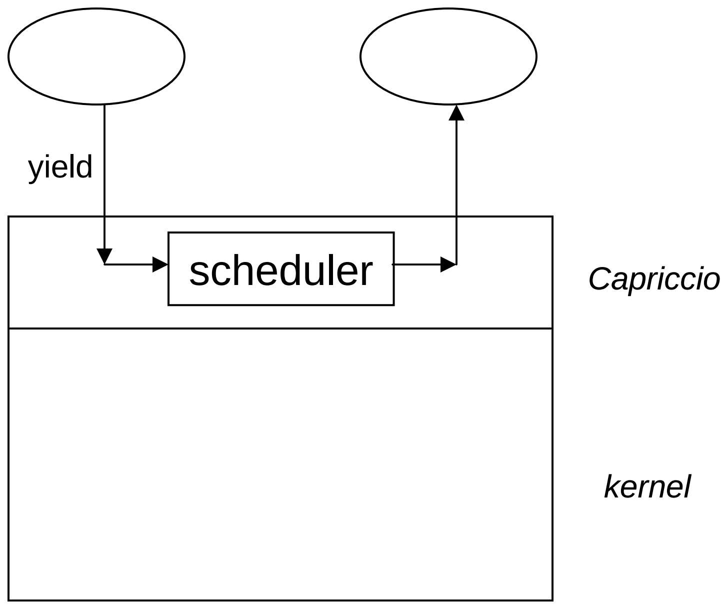
- **Seda**
  - Each thread bound to one stage of the computation
  - Each service request proceeds through successive stages

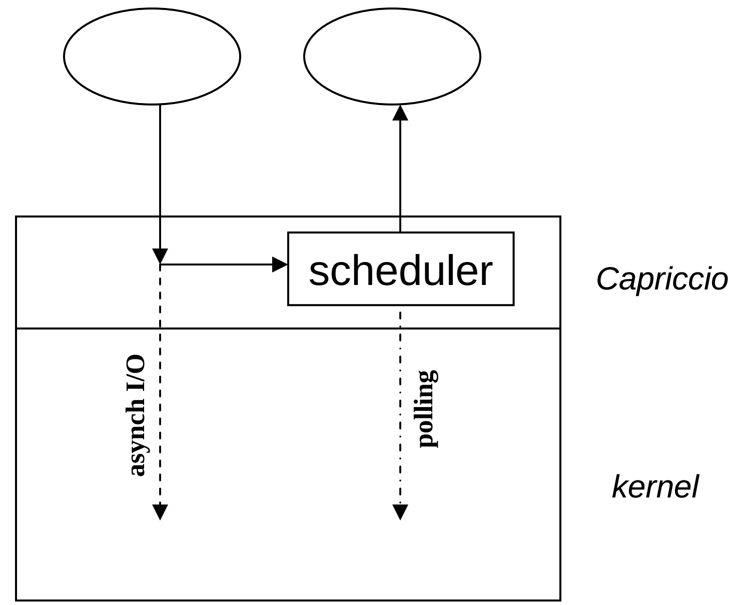


Capriccio

- Philosophy
  - Thread *model* is useful
  - Improve *implementation* to remove barriers to scalability
- Techniques
  - User-level threads
  - Linked stack management
  - Resource aware scheduling
- Tools
  - Compiler-analysis
  - Run-time monitoring

Capriccio – user level threads

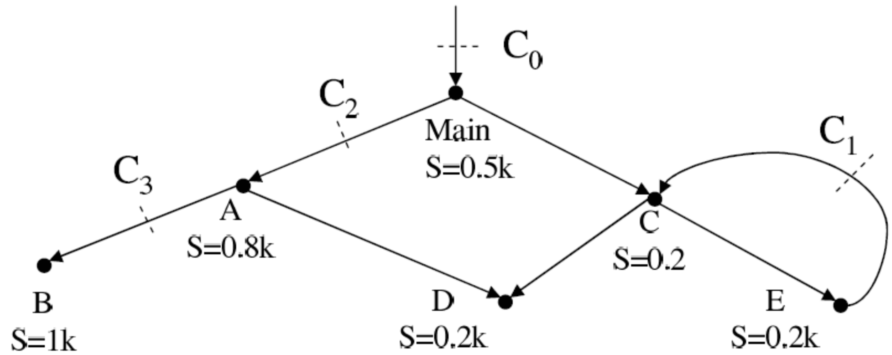




- User-level threading with fast context switch
- Cooperative scheduling (via yielding)
- Thread management costs independent of number of threads (except for sleep queue)

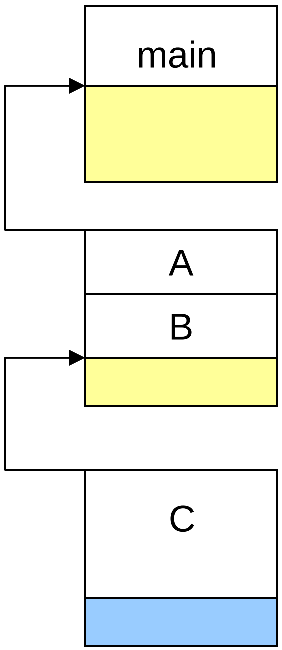
- Intercepts and converts blocking I/O into asynchronous I/O
- Does polling to determine I/O completion

Compiler Analysis - Checkpoints



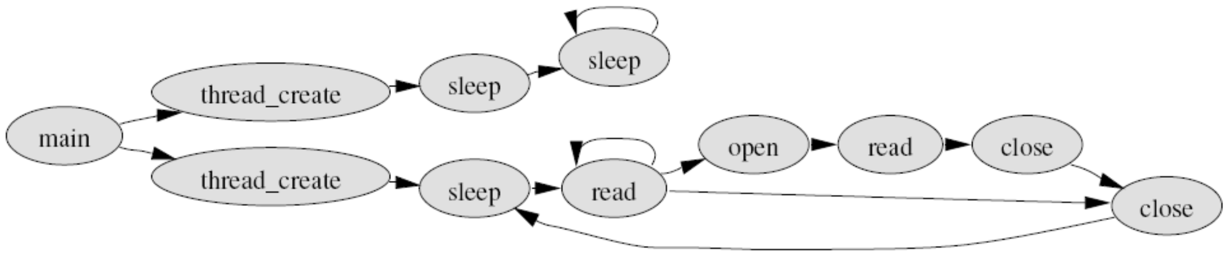
- Call graph – each node is a procedure annotated with maximum stack size needed to execute that procedure; each edge represents a call
- Maximum stack size for thread executing call graph cannot be determined statically
  - Recursion (cycles in graph)
  - Sub-optimal allocation (different paths may require substantially different stack sizes)
- Insert checkpoints to allocate additional stack space ("chunk") dynamically
  - On entry (e.g., C_O)
  - On each back-edge (e.g. C_1)
  - On each edge where the needed (maximum) stack space to reach a leaf node or the next checkpoints exceeds a given limit (*MaxPath*) (e.g., C_2 and C_3 if limit is 1KB)
- Checkpoint code added by source-source translation

Linked Stacks



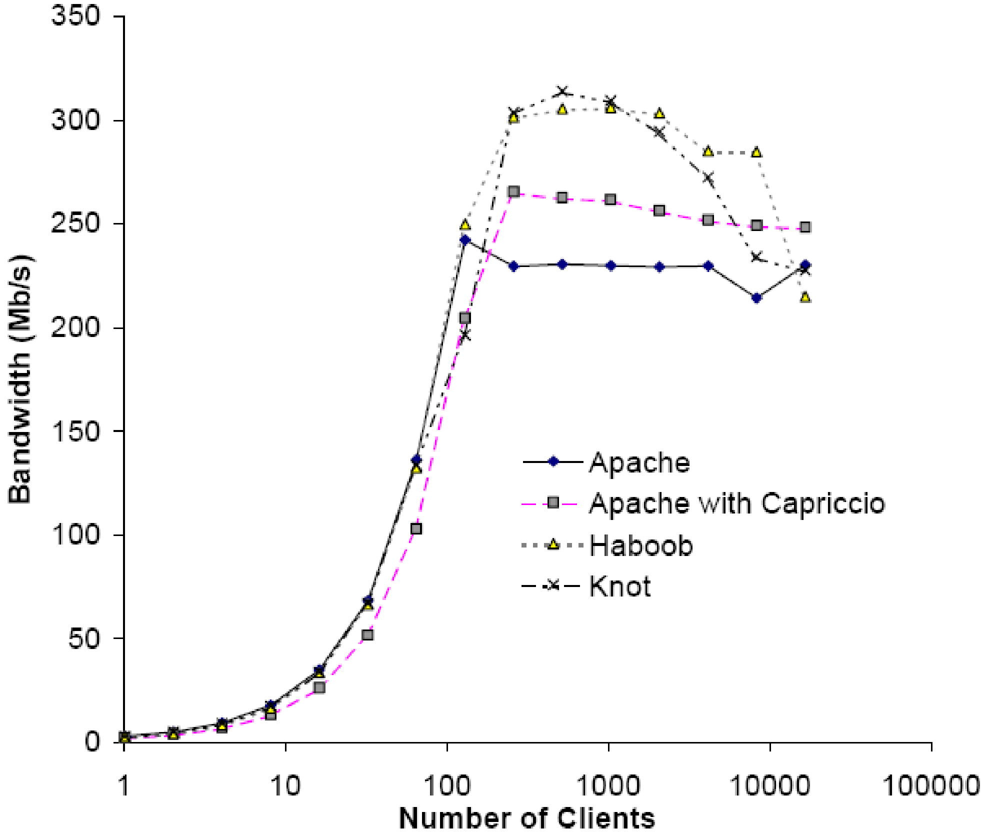
- Thread stack is collection of non-contiguous blocks (‘chunks”)
- *MinChunk:* smallest stack block allocated
- Stack blocks “linked” by saving stack pointer for “old” block in field of “new” block; frame pointer remains unchanged
- Two kinds of wasted memory
  - Internal (within a block) (yellow)
  - External (in last block) (blue)
- Two controlling parameters
  - MaxPath: tradeoff between amount of instrumentation and run-time overhead vs. internal memory waste
  - MinChunk: tradeoff between internal memory waste and external memory waste
- Memory advantages
  - Avoids pre-allocation of large stacks
  - Improves paging behavior by (1) leveraging LIFO stack usage pattern to share chunks among threads and (2) placing multiple chunks on the same page

Resource-aware scheduling



- Blocking graph
  - Nodes are points where the program blocks
  - Arcs connect successive blocking points
- Blocking graph formed dynamically
  - Appropriate for long-running program (e.g. web servers)
- Scheduling annotations
  - Edge – exponentially weighted average resource usage
  - Node – weighted average of its edge values (average resource usage of next edge)
  - Resources – CPU, memory, stack, sockets
- Resource-aware scheduling:
  - Dynamically prioritize nodes/threads based on whether the thread will increase or decrease its use of each resource
  - When a resource is scarce, schedule threads that release that resource
- Limitations
  - Difficult to determine the maximum capacity of a resource
  - Application-managed resources cannot be seen
  - Applications that do not yield

Performance comparison



- Apache – standard distribution
- Haboob – event-based web server
- Knot – simple, threaded specially developed web server