HL++ 2022 HighLoad++ FOUNDATION

Яндекс

Эволюция акторной системы

Алексей Станкевичус,
руководитель группы разработки

Содержание

01

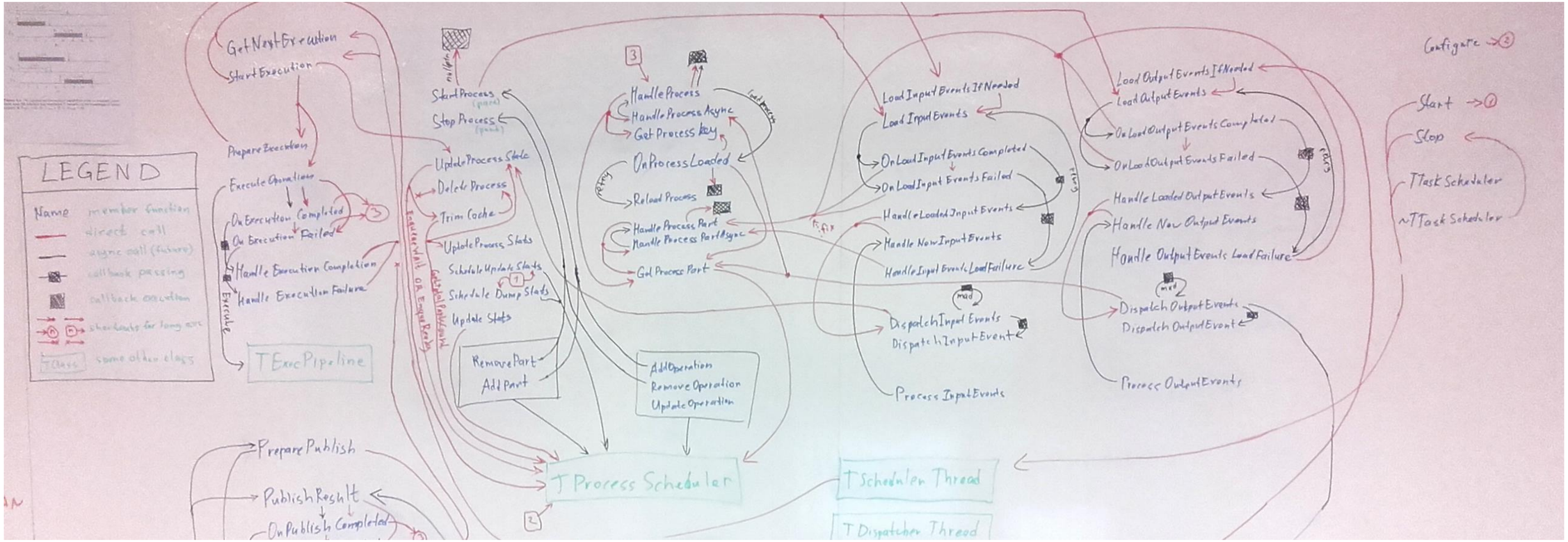
Краткий обзор

подходы к созданию
многопоточных программ на C++

Подходы к созданию многопоточных программ

- Разделяемая память (mutex, condvar, future-promise)
- Передача сообщений
- Гибридный способ

Взаимодействие через разделяемую память

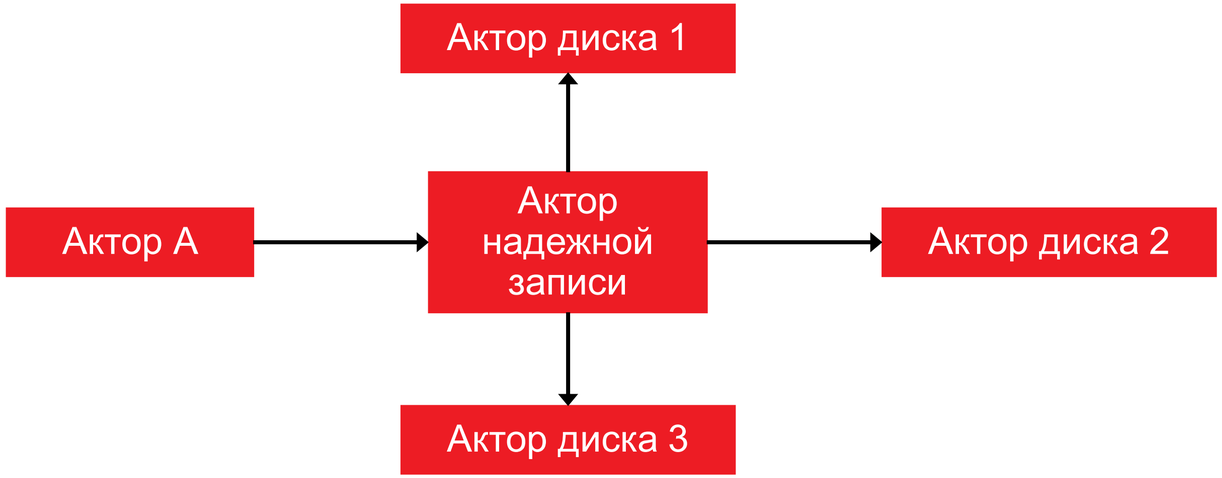


Передача сообщений: модель акторов

Актор является конечным автоматом, а любое взаимодействие — передачей сообщения, что помогает структурировать код и снижает сложность по сравнению с альтернативными подходами.

Легко переносится из плоскости «параллельного» программирования в плоскость «распределенного». Акторы позволяют сделать взаимодействие между хостами прозрачным.

В YDB мы выбрали использовать модель акторов и с нуля создали свою акторную систему. С тех пор прошло более 7 лет, и сегодня акторная система исполняется на десятках тысяч серверов.



Исходный код

Исходный код YDB доступен под лицензией Apache 2.0,
акторная система лежит в основе YDB

Код акторной системы находится по адресу
https://github.com/ydb-platform/ydb/tree/main/library/cpp/actors

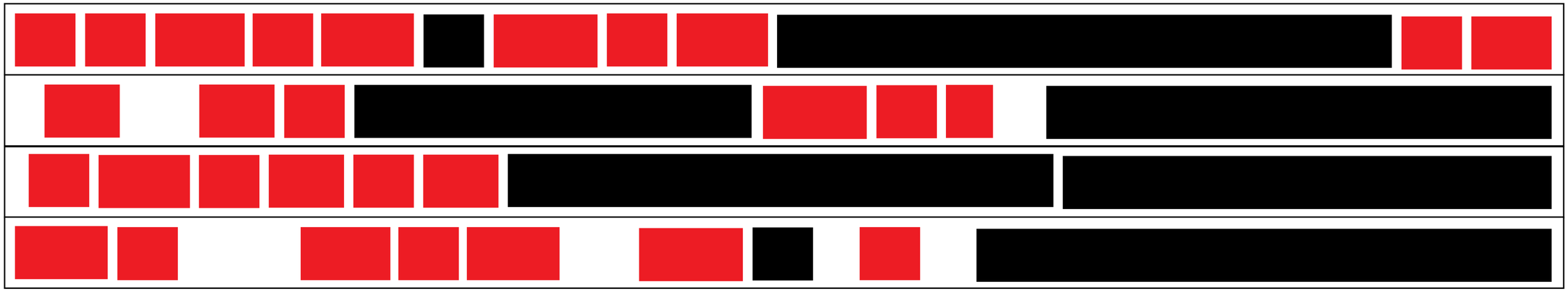
02

Проблема фоновой нагрузки

фоновые задачи портят latency
интерактивной нагрузки

Проблема фоновой нагрузки

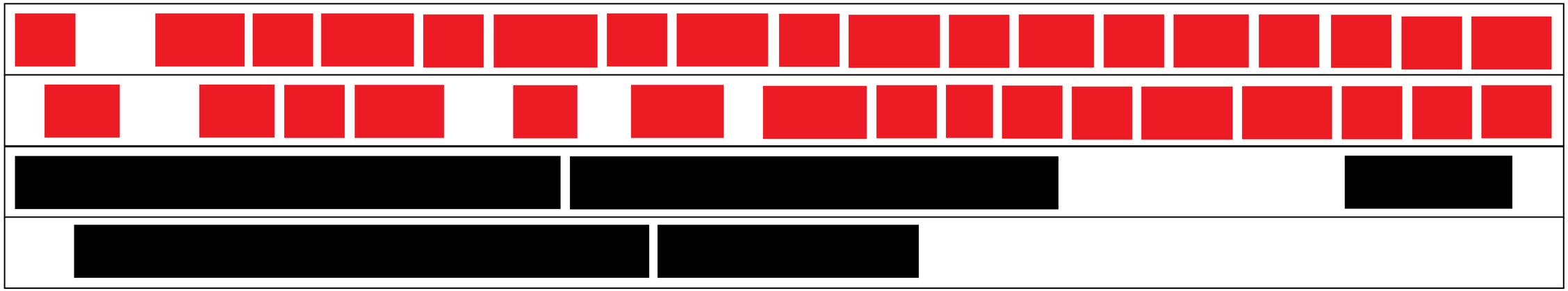
В YDB нагрузку можно разделить на две части. Интерактивная — обработка запросов от пользователя, где latency имеет значение. Фоновая — например, compaction LSM дерева. Фоновые задачи могут требовать много CPU и надолго непрерывно занимать потоки исполнения, ухудшая latency интерактивной нагрузки. Как совместить интерактивную нагрузку и фоновые задачи в одном приложении?



Пулы потоков

Простое решение проблемы — ввести отдельные пулы потоков для разных классов нагрузки. Позволяет изолировать интерактивные задачи от фоновых. Минусы подхода — резервирование CPU под разные классы задач и ограничение утилизации.

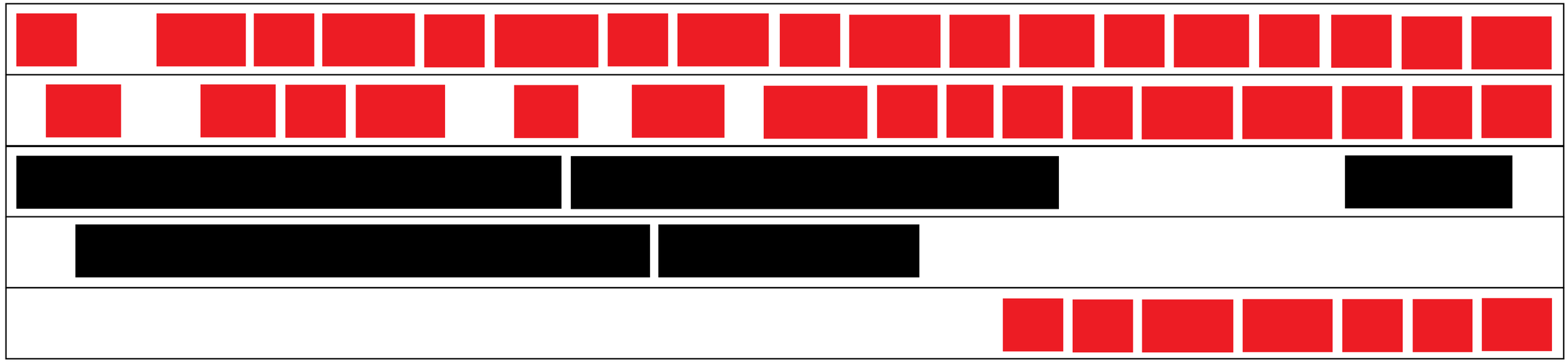
Actor System 1.0: акторы живут в пулах с настраиваемым количеством потоков.



Пулы потоков с переподпиской

Загрузка пулов неравномерна, возникает желание иметь в пулах больше потоков, чем есть ядер у машины.

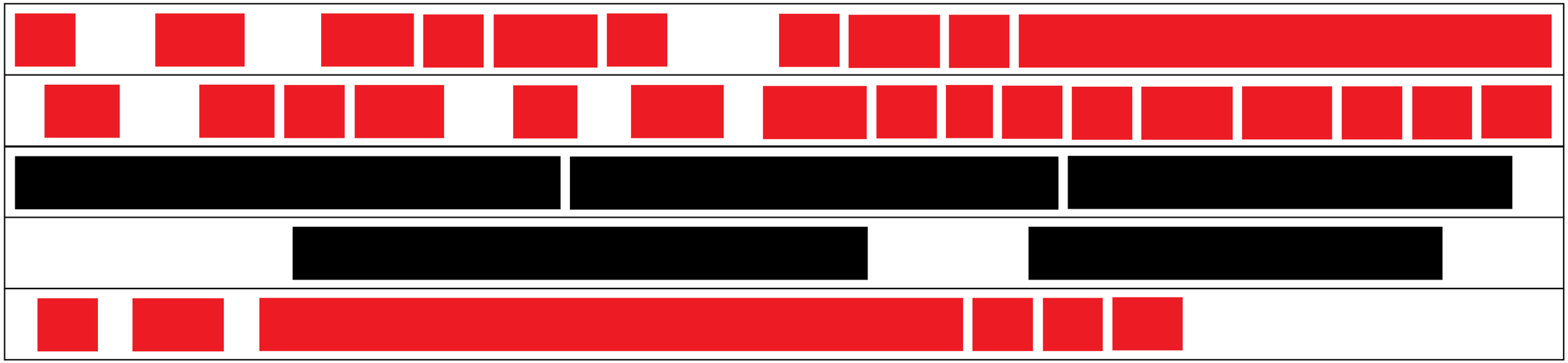
Ожидание:



Пулы потоков с переподпиской

Реальность.

Вытеснение CFS–планировщиком Linux. Нельзя просто полагаться на планировщик потоков ОС в случае переподписки, поскольку он вытесняет случайный актор в середине исполнения, блокируя его на неопределенное время и ухудшая latency.



03

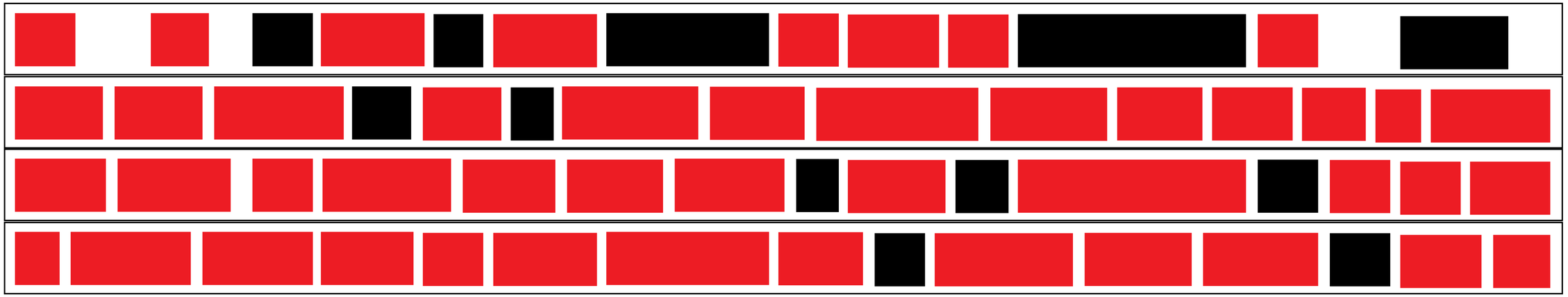
Real-time, watchdog и таймеры

Actor System 2.0a

Эволюция

Actor System 1.0: акторы живут в пулах с настраиваемым количеством потоков

Actor System 2.0a: вытесняющая многозадачность в userspace



Можно ли реализовать вытеснение в userspace?

Как сделать эффективное управление вытеснением потоков с накладными расходами <1%

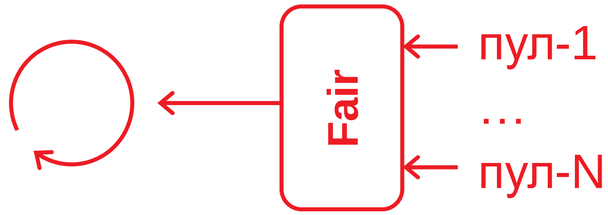
Потоки на каждом ядре

RT 0

watchdog

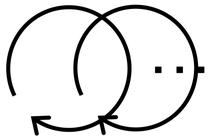
Watchdog занимается вытеснением потоков
с долго обрабатывающими сообщение акторами

RT 1



RT–поток пулов обеспечивает возможность обработать сообщения для акторов из всех пулов каждые TargetLatency (1 миллисекунда)

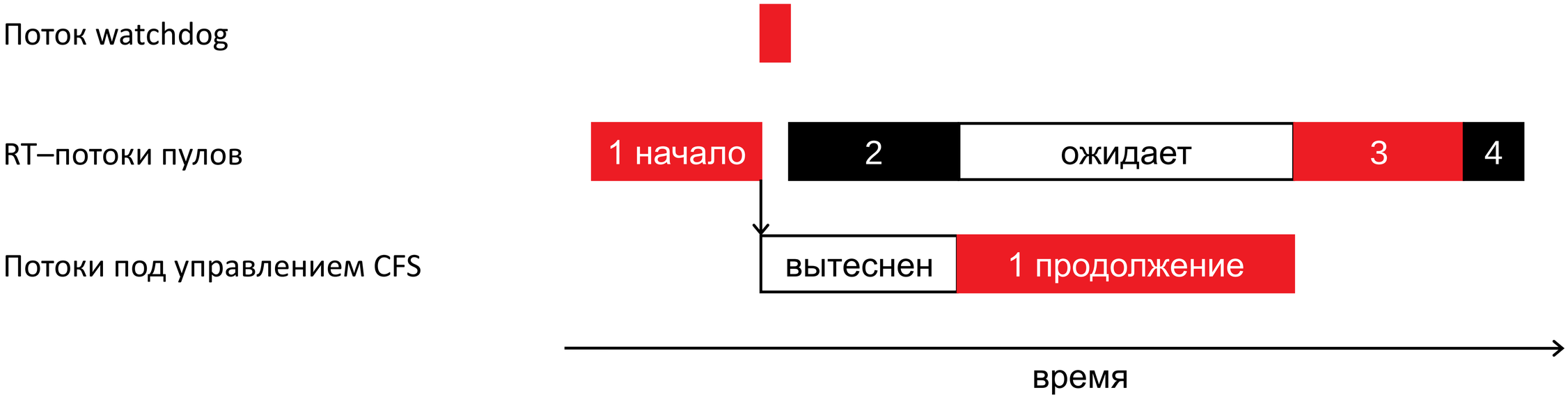
CFS



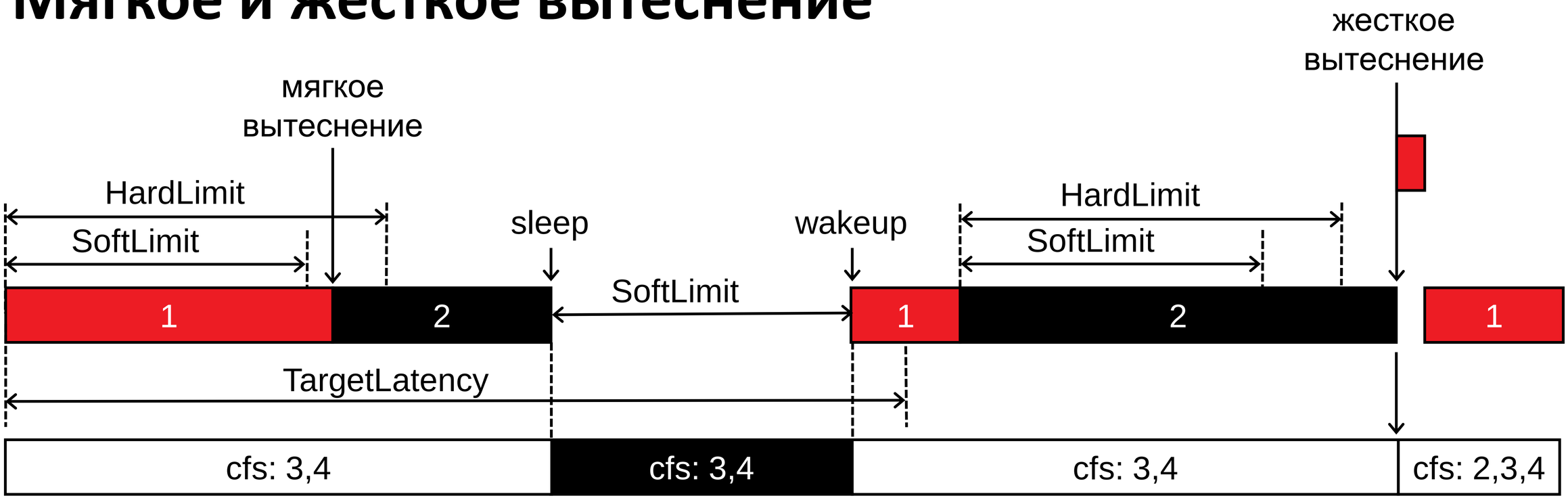
пул-1 … пул-N

Потоки под управлением CFS получают возможность выполняться когда RT–поток
пулов спит

Вытеснение актора сменой приоритета



Мягкое и жесткое вытеснение



Пытаемся запускать акторы каждого пула не реже, чем раз в TargetLatency (1ms)

- Мягкое вытеснение через SoftLimit = TargetLatency / ActivePoolCount
- Жесткое вытеснение через HardLimit = SoftLimit + EventLimit (100us)

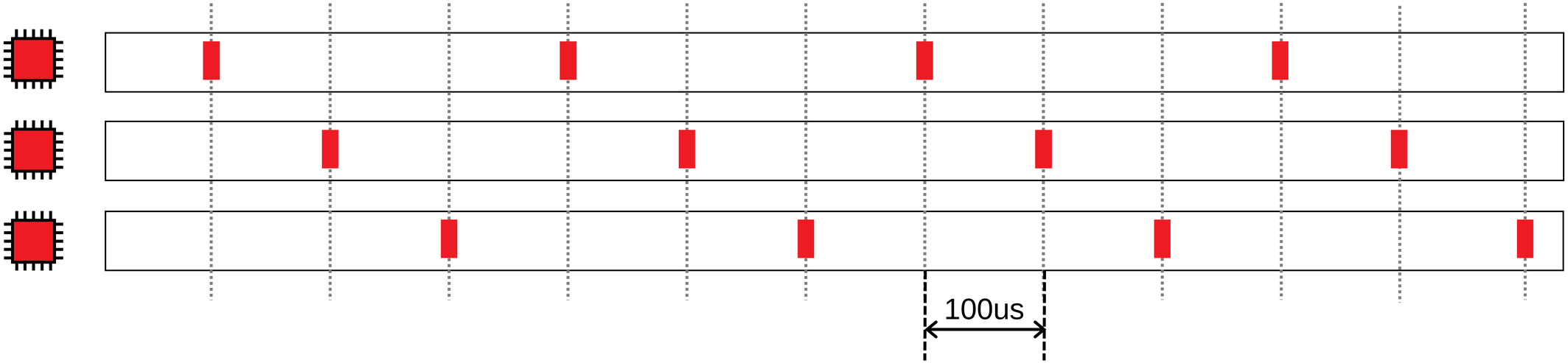
Watchdog

Watchdog просыпается по расписанию и изменяет приоритеты рабочих потоков

Потоки закреплены на ядрах с помощью affinity

Недостаток — постоянные переключения контекста

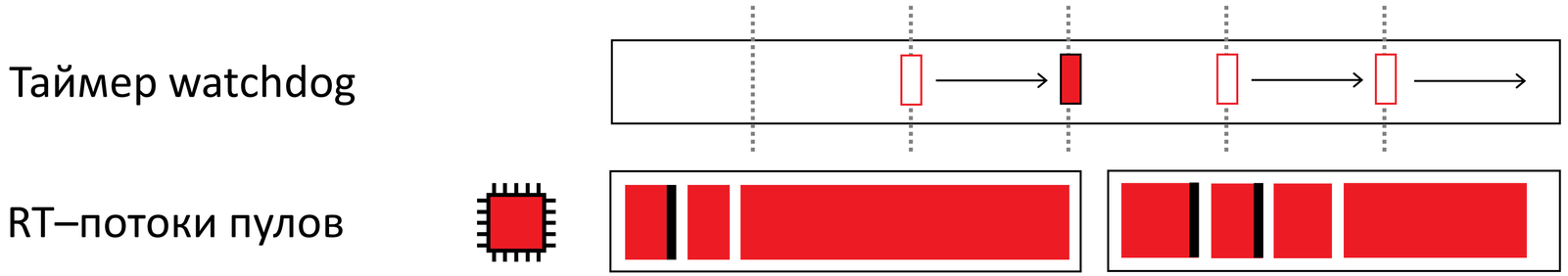
Накладные расходы 3-4% при latency <1ms и 100% утилизации



\+ Watchdog просыпается в разных потоках, но это один watchdog, он занимается вытеснением любых rt-потоков

Таймеры

Чтобы избавиться от дорогих переключений контекста, мы использовали таймеры. Перед тем как начать исполнение задачи, поток устанавливает таймер. Если задача укладывается в отведенное время, таймер перезапускается для другой задачи. Если же задача занимает слишком много времени, мешая другим — срабатывает таймер и происходит вытеснение. Причем таймер срабатывает не в специальном потоке, а в обычном рабочем потоке, который сразу же начинает исполнение другой задачи. При таком подходе все потоки одинаковы, а вытеснение требует ровно 1 переключение контекста.



Проблемы

Работа драйверов и других программ

Высокоприоритетный поток не дает выполняться другим программам, в том числе может мешать обработке отложенных прерываний.

Запуск 2 и более приложений на машине

Нужно согласованно разделять ядра между приложениями, использующими realtime–приоритет.

Эволюция

Actor System 1.0: акторы живут в пулах с настраиваемым количеством потоков

~~Actor System 2.0a: вытесняющая многозадачность в userspace~~

Actor System 1.5: обмен потоками

Actor System 1.75: передача 0.5 потока

Actor System 2.0b: ядро-вытрезвитель

04

Обмен потоками

потоки можно передавать из
свободного пула
в перегруженный

Нельзя просто так взять и перекинуть поток из пула в пул

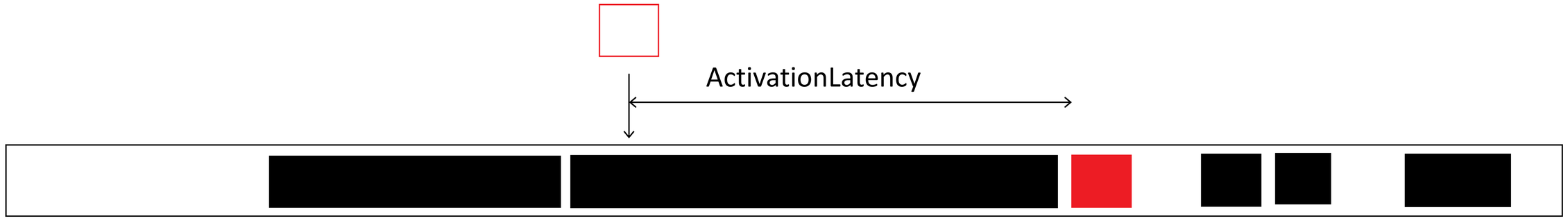
Насколько все уже плохо?

Каждый pool собирает статистику использования за некоторый период времени:

— CpuLoad: среднее потребление, в процессорах, CpuIdle = CurrentCpus - CpuLoad

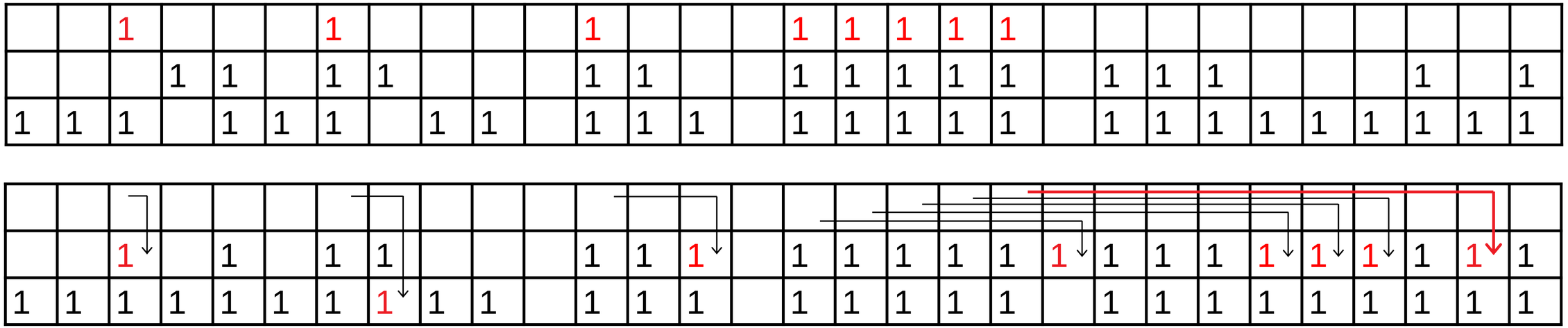
— ActivationLatency: худшее фактическое время активации за последний период

— Load1Latency: оценочная длительность busy period при условии, что пул отдал 1 поток (для оценки worst case latency)

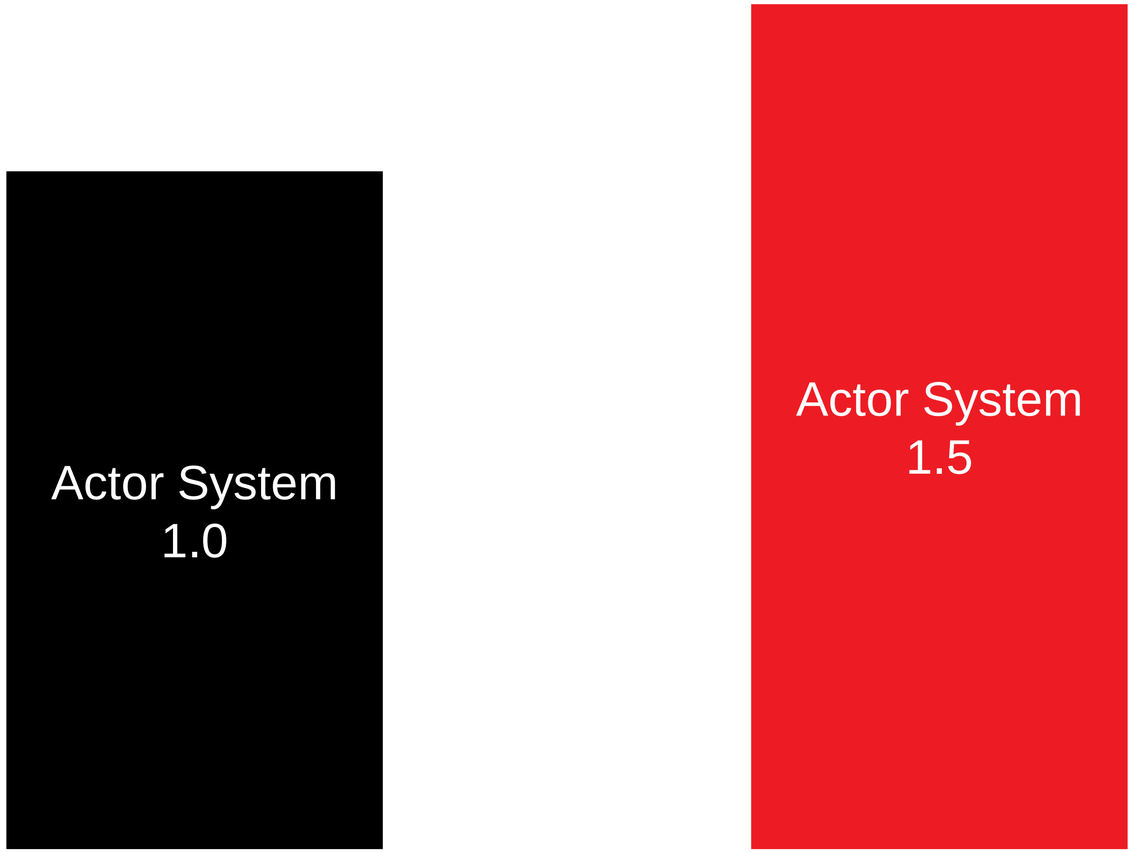


Насколько станет хуже без 1 потока?

- Оценка сверху
- Циклический буфер TimeSeries с сеткой 10 микросекунд
- Wait-free
- Load1Latency вычисляется при постобработке
- ActivationLatency + Load1Latency сравнивается с заданным в настройках порогом



Производительность на том же железе



05

0.5 ядра (но это не точно) и вытрезвитель

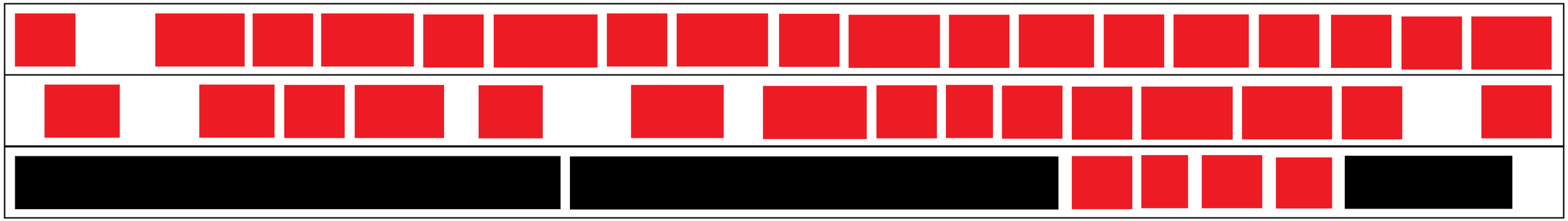
повышаем гранулярность
передачи ядер с помощью
невытесняющей многозадачности

Дробное количество ядер

Разрешать делить 1 поток двум пулам поровну, без жесткого вытеснения.

Не допускать ситуации, когда 1 пул имеет 2 половинки ядра вместо одного целого, балансировщик должен склеивать такие ядра в целое ядро.

Это будет лучше, чем переподписка, и будет меньше стробить при перегрузке части пулов.
Можно будет оставить менее 1 ядра в пуле.



Эволюция

Actor System 1.0: акторы живут в пулах с настраиваемым количеством потоков

~~Actor System 2.0a: вытесняющая многозадачность в userspace~~

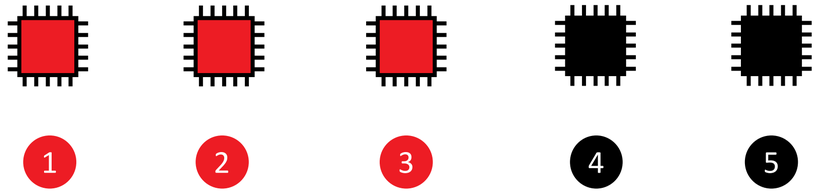
Actor System 1.5: обмен потоками

Actor System 1.75: передача 0.5 потока

Actor System 2.0b: ядро-вытрезвитель

Ядро–вытрезвитель

Ядра для хороших потоков



Ядро-вытрезвитель
для потоков-хулиганов

HighLoad++
FOUNDATION

Яндекс

Спасибо!

Алексей Станкевичус,

руководитель группы разработки

the_ancient_one@aol.com