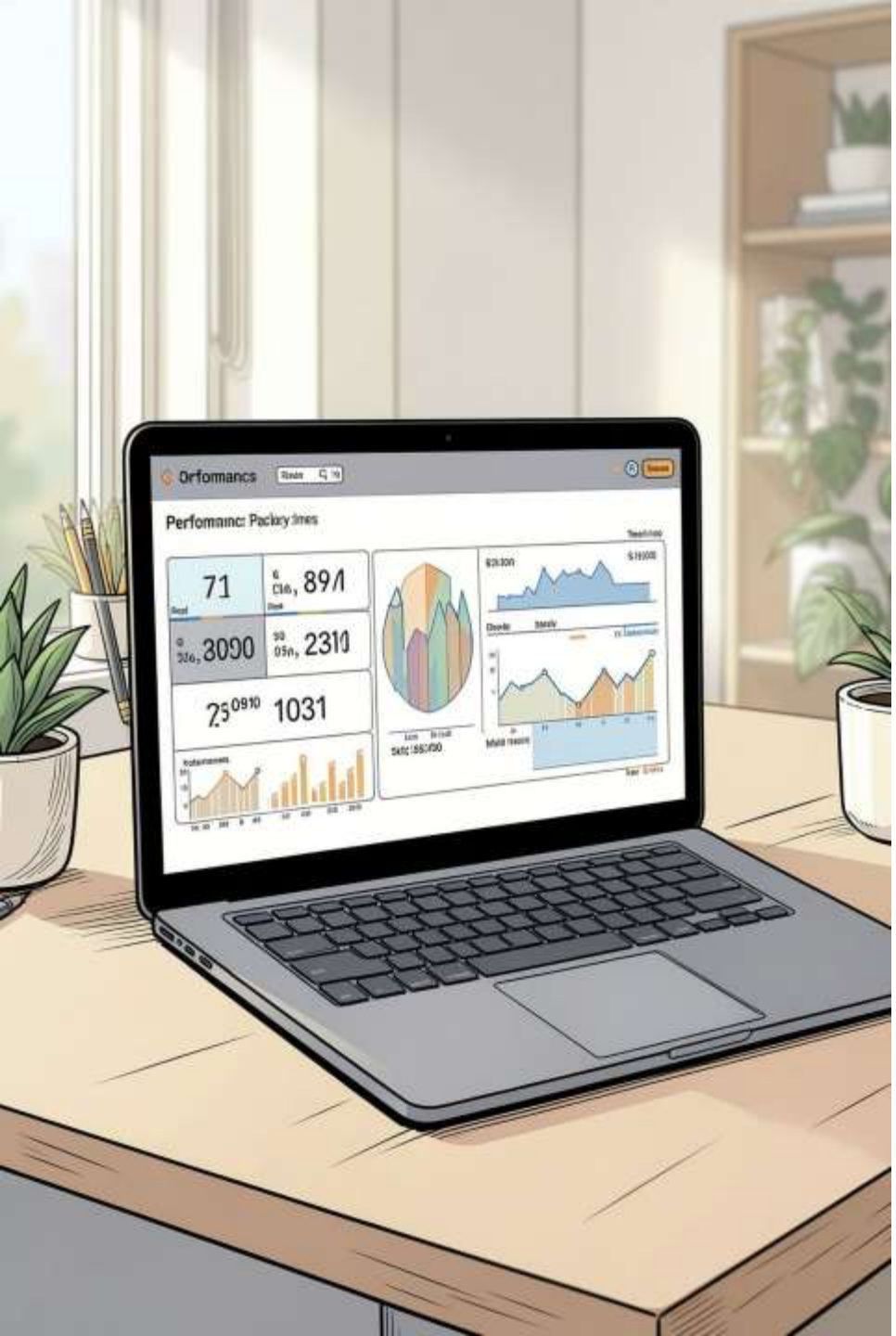
АО «АЛМАТИНСКИЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»
https://atu.edu.kz/

Факультет «Информационные технологии»
Кафедра «Компьютерная инженерия»

Название курса: «Инструментальные средства разработки программ»



Зиятбекова Гулзат Зиятбеккызы – PhD, ассоциированный профессор
g.ziyatbekova@atu.edu.kz



Оптимизация времени обработки и размера: Инструменты и методы

Краткое введение: эта презентация описывает принципы и практики, которые помогают снизить время выполнения задач и уменьшить размер артефактов. Мы пройдем от диагностики и поиска узких мест до конкретных техник — оптимизации алгоритмов, структур данных, кэширования, асинхронности и инструментов профилирования. Цветовой акцент: тёмный — #1D1D1B, фон — #EFEFEF.

Почему оптимизация важна? Проблемы и преимущества

Проблемы

Высокая латентность, перерасход CPU/RAM, долгие сборки, большие бинарники, ухудшение UX и рост расходов на инфраструктуру.

Преимущества

Быстрая отдача пользователю, снижение затрат, масштабируемость, лучшее время отклика и конкурентное преимущество.

Ограничения и риск

Прематчинг оптимизаций: преждевременная оптимизация усложняет код, повышает хрупкость и затрудняет поддержку — применять целенаправленно.



Анализ производительности: выявление узких мест

Цель: собрать фактические данные перед оптимизацией. Основные этапы: инструментальное профилирование (CPU, память, I/O), трассировка времени отклика, метрики на проде (APM), и тесты нагрузки. Результат — измеримая карта узких мест с приоритетами по влиянию на время и размер.

1 Сбор данных

Логи, метрики, трассировки распределённой системы, снапшоты памяти.

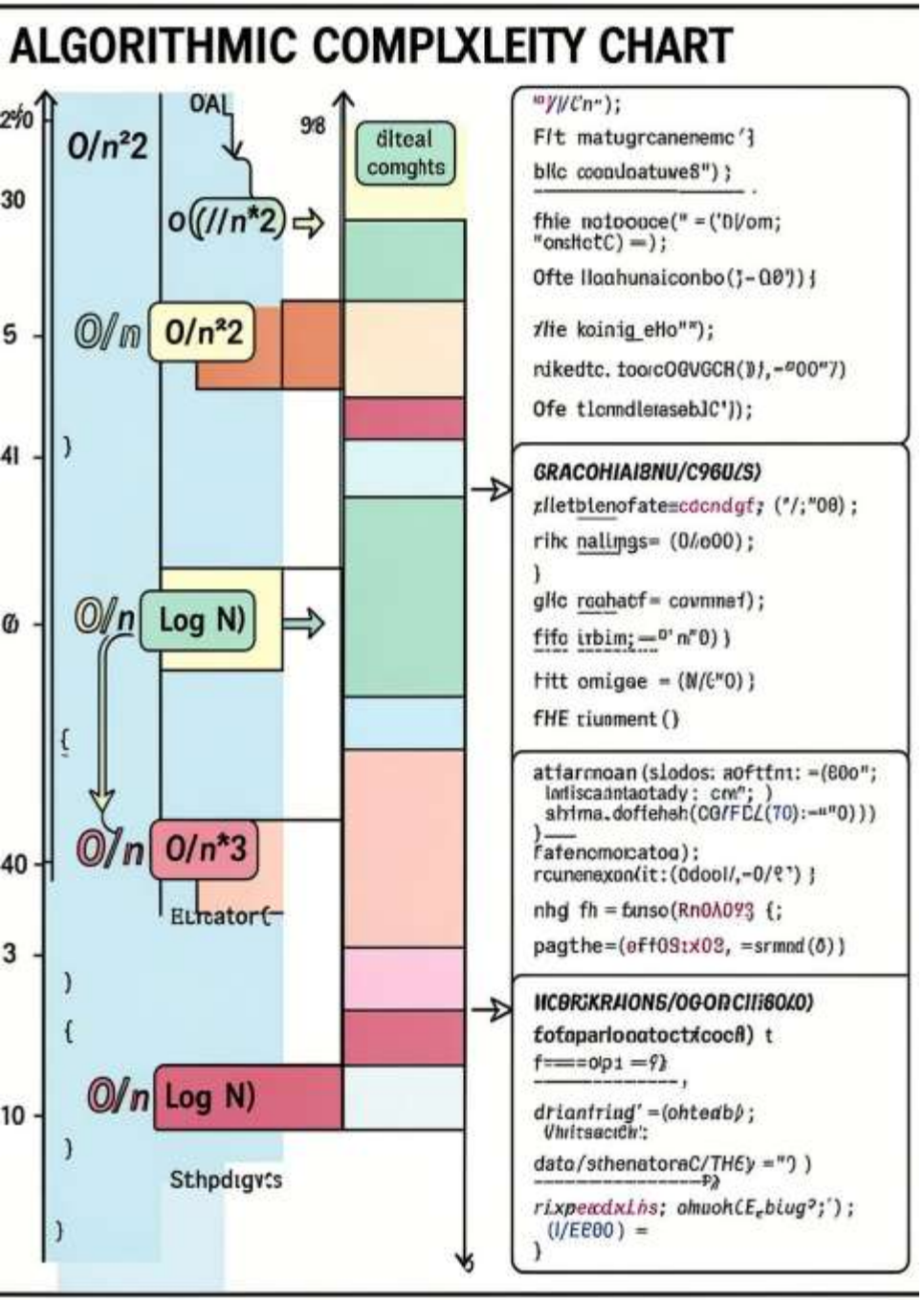
2 Анализ

Определить горячие функции, горячие запросы и узкие места ввода-вывода.

3 Приоритизация

Оценить возврат инвестиций (RTI): время/размер vs усилия на оптимизацию.





Методы оптимизации алгоритмов: эффективность кода

Фокус на алгоритмической сложности: выбрать лучшие алгоритмы и подходы, которые уменьшают асимптотику. Частые шаги — заменять O(n^2) на O(n log n) или O(n), применять жадные, разделяй-и-властвуй, индексирование и предварительную сортировку. Важно: измерять изменения до и после.

Паттерны

- Разделяй и властвуй (merge/quick sort)
- Двухуказательный подход
- Поиск и индексирование (hash / B-tree)

Оптимизация горячих путей

Минимизируйте allocations, избегайте частых аллокаций в циклах, инлайнинг критичных функций и использование эффективных итераторов.

Оптимизация структур данных: выбор правильного инструмента

Правильная структура данных часто важнее микрооптимизаций. Выбирайте по требованиям: быстрый доступ — массив/хэш; частые вставки — связный список или deque; упорядоченные данные — сбалансированное дерево. Также учитывать память: компактные представления и структурные сжатия (bit-packing, sparse structures).

Хэш-таблицы

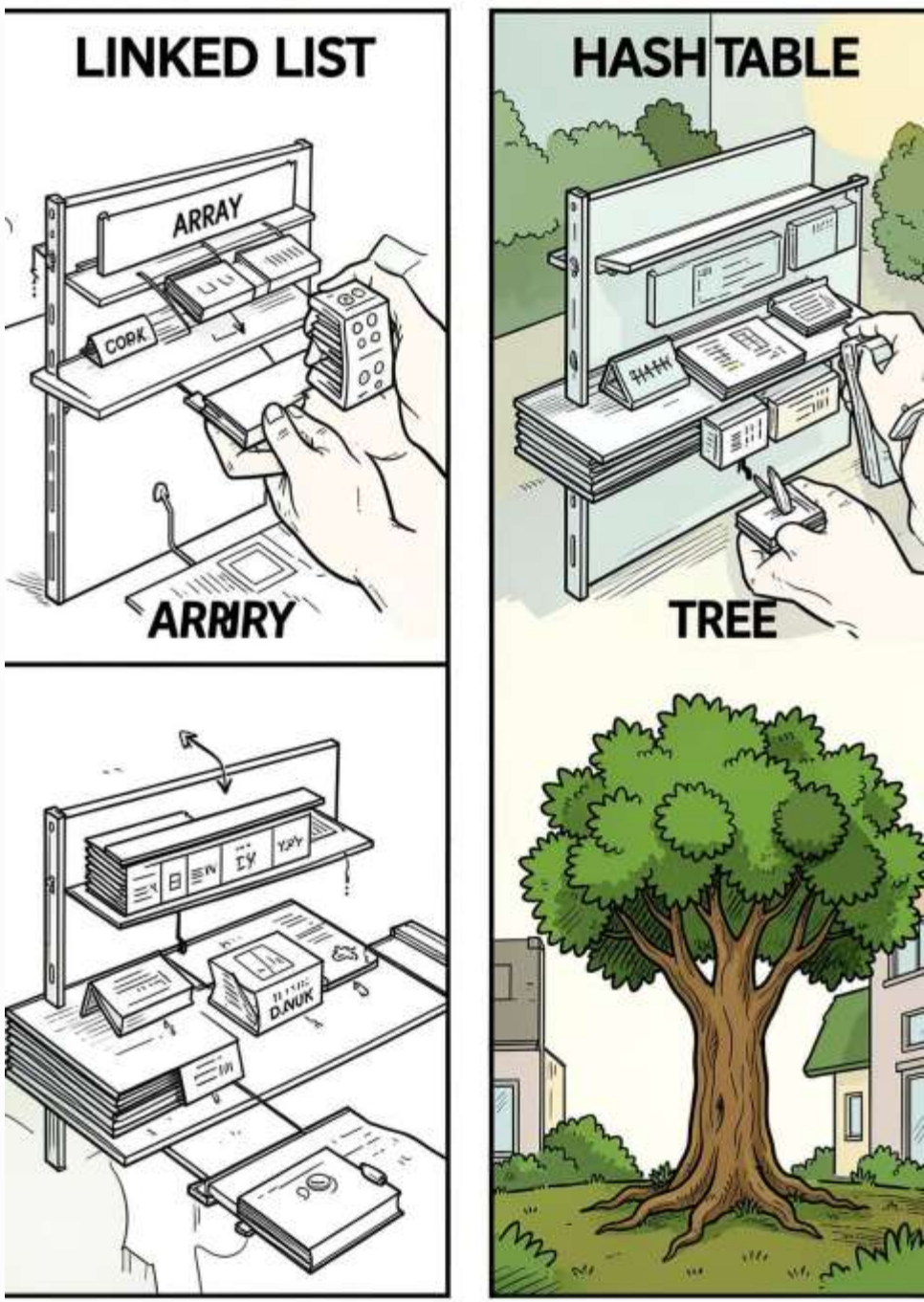
O(1) доступ — идеальны для быстрых поисков, но потребляют память и чувствительны к коллизиям.

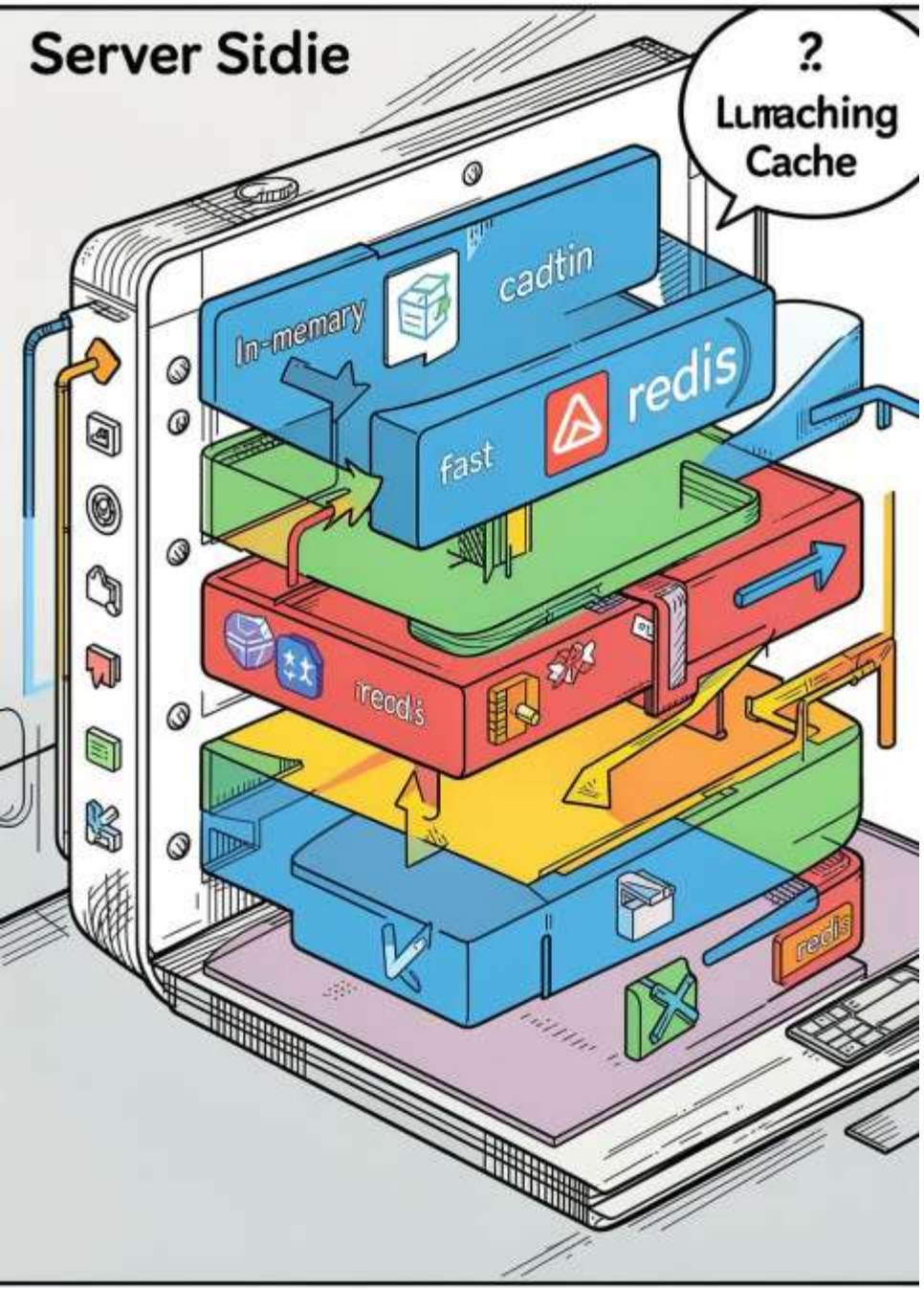
Деревья

O(log n) для поиска/вставки, упорядоченные обходы и диапазонные запросы.

Массивы

Компактная память и высокая локальность кэша — хороши для последовательного доступа.





Кэширование и мемоизация: ускорение повторяющихся операций

Кэширование уменьшает повторную работу: использовать L1 (память процесса), L2 (Redis/Memcached) и CDN для статики. Мемоизация подходит для чистых функций. Обратить внимание на стратегию инвалидации, размер кэша, TTL и согласованность данных.

Типы кешей

- Локальный (in-process)
- Распределённый (Redis)
- Кэш уровня сети (CDN)

Практика

Кэшировать результаты дорогих вычислений, агрегатов, часто запрашиваемых ресурсов. Тестировать эффект и мониторить промахи (cache misses).



Асинхронное программирование: улучшение отзывчивости

Асинхронность позволяет не блокировать потоки на I/O-операциях. Модели: коллбэки, промисы/futures, async/await и реактивные потоки. Применять для сетевых операций, доступа к БД и файловых операций. Следить за ограничением параллелизма (semaphores, worker pools) и утечками задач.

Паттерны

Async/await, non-blocking I/O, backpressure в потоках данных.

Контроль параллелизма

Ограничение числа одновременных задач, пул потоков, очередь задач.

Оптимизация размера кода: сжатие и минификация

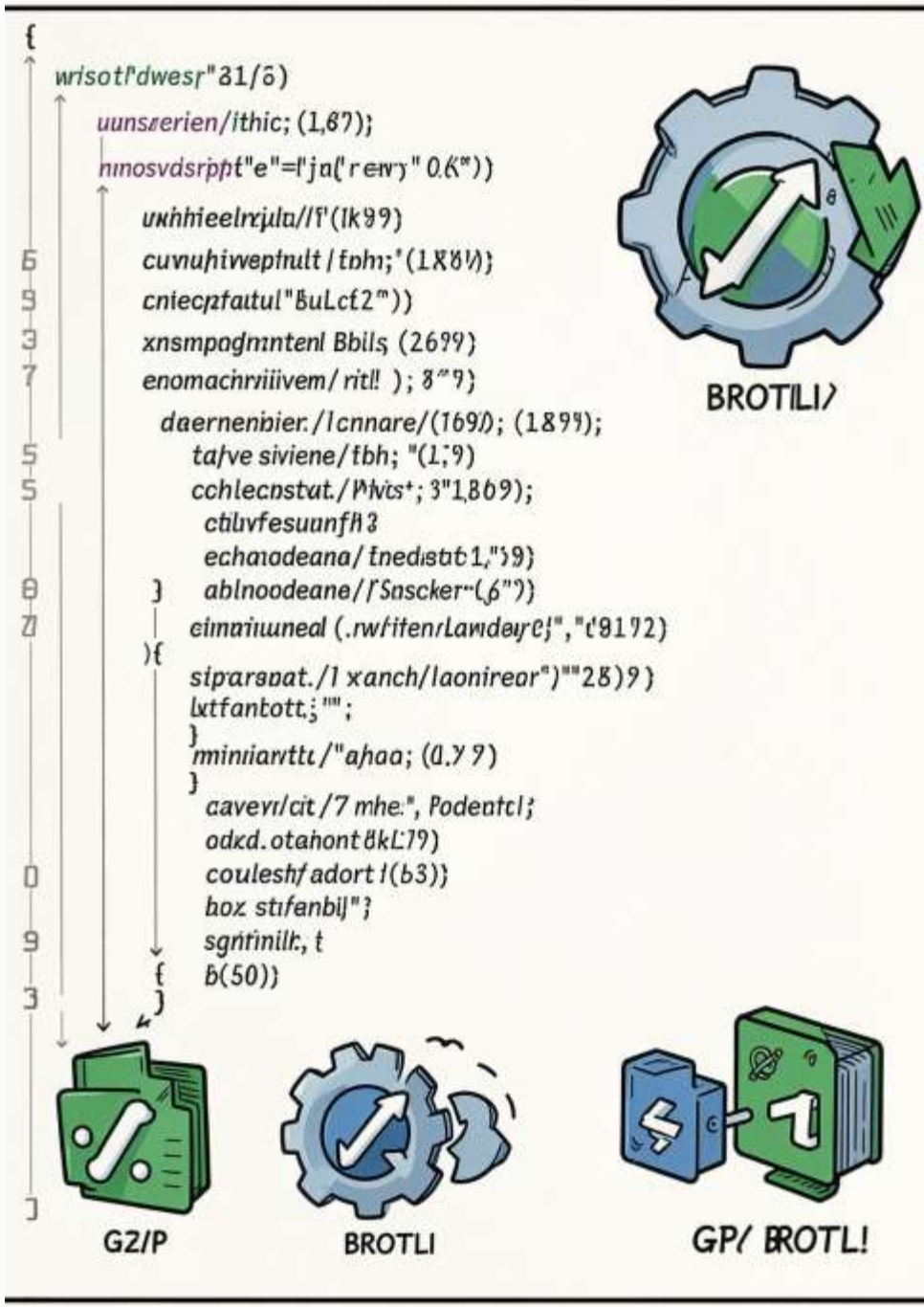
Снижение размеров артефактов уменьшает время передачи и ускоряет загрузку. Инструменты: минификаторы (Terser, Uglify), tree-shaking (ec6 imports), сжатие на уровне сервера (gzip, Brotli), бинарное сжатие (upx для исполняемых файлов) и разделение кода (code-splitting). Контролируйте влияние на отладку и используйте source maps.

Frontend

Tree-shaking, lazy-loading, CSS-пурдж, сжатие контента и оптимизация изображений.

Backend и бинарники

Удаление ненужных зависимостей, статическая линковка по необходимости, бинарные упаковщики, сжатие распределённых образов.



Инструменты для анализа и профилирования производительности

Категории инструментов: системный профайлинг (perf, pprof), анализ памяти (heap profilers, Valgrind), APM (New Relic, Datadog), браузерные devtools, нагрузочные тесты (JMeter, k6), статический анализ и сборщики (Webpack Bundle Analyzer). Выбирайте по цели: latency, CPU, memory или bundle size.

Профилирование **CPU**

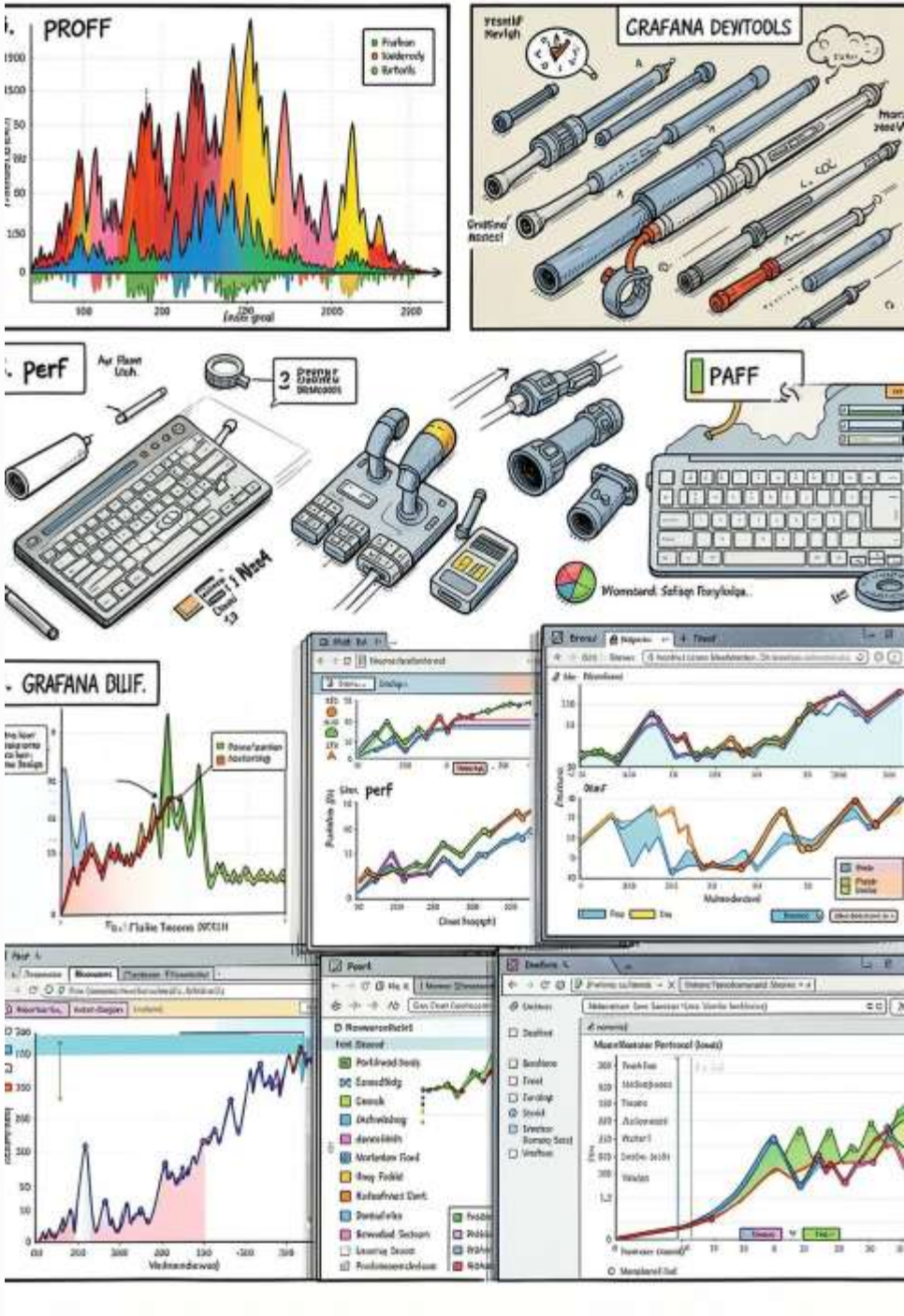
Flame graphs, pprof — показывают горячие функции и их вклад во время выполнения.

Анализ памяти

Heap snapshots, leak detectors — выявляют удерживаемые объекты и утечки.

Мониторинг

APM и метрики (Prometheus/Grafana) для наблюдения в проде и быстрого реагирования.



Заключение: лучшие практики и дальнейшие шаги

Резюме рекомендаций: начинайте с метрик и профайлинга, приоритизируйте воздействия, выбирайте алгоритмы и структуры данных по задаче, применяйте кэширование и асинхронность там, где это эффективно, и уменьшайте размер артефактов с помощью сборщиков и сжатия. Регулярно прогоняйте тесты производительности и автотесты регрессии.

План действий

1. Собрать метрики и профайлы
2. Приоритизировать узкие места
3. Внедрить оптимизации по шагам
4. Замерить эффект и зафиксировать PR

Контроль качества

- Регрессионные тесты
- Мониторинг SLA
- Документирование изменений

Совет: избегайте преждевременной оптимизации — фокусируйтесь на измеримых выигрышах и поддерживаемом коде.

