Effect

Sebastian Lorenz @thefubhy https://github.com/fubhy


EFFECT
EFFECT.
EFFECT!!
EFFECT
EFFECT
EFFECT
EFFECT
EFFECT
EFFECT
EFFECT
EFFECT
EFFECT
EFFECT
EFFECT
EFFECT
EFFECT
EFFECT
EFFECT
EFFECT
EFFECT

Our “Simple” Program

```
const fetchDocument = async (id: number): Promise<unknown> => {
  const response = await fetch(`https://hacker-news.firebaseio.com/v0/item/${id}.json`)
  return await response.json()
}

const fetchDocuments = async (ids: ReadonlyArray<number>): Promise<Array<unknown>> => {
  const documents: Array<unknown> = []
  for (const id of ids) {
    documents.push(await fetchDocument(id))
  }
  return documents
}

const main = async () => {
  const documents = await fetchDocuments([1, 2, 3, 4, 5])
  for (const document of documents) {
    console.log(`Document: ${JSON.stringify(document)}`)
  }
}
```

No Concurrency

```ts
const fetchDocuments = async (ids: ReadonlyArray<number>): Promise<Array<unknown>> => {
  const documents: Array<unknown> = []
  for (const id of ids) {
    documents.push(await fetchDocument(id))
  }
  return documents
}
```

Unbounded Concurrency

```
const fetchDocuments = async (ids: ReadonlyArray<number>): Promise<Array<unknown>> ⇒ {
  const documents: Array<Promise<unknown>> = []
  for (const id of ids) {
    documents.push(fetchDocument(id))
  }
  return await Promise.all(documents)
}
```

Naive Controlled Concurrency

```ts
const fetchDocuments = async (ids: ReadonlyArray<number>, concurrency: number): Promise<Array<unknown>> => {
  const documents: Array<unknown> = []
  for (let i = 0; i < ids.length; i += concurrency) {
    const chunk = ids.slice(i, i + concurrency).map((_) => fetchDocument(_))
    documents.push(...(await Promise.all(chunk)))
  }
  return documents
}
```

Full Controlled Concurrency

```ts
const fetchDocuments = async (ids: ReadonlyArray<number>, concurrency: number): Promise<Array<unknown>> => {
  const remaining = ids.slice().map((id, index) => [id, index] as const).reverse()
  const documents: Array<unknown> = []
  return new Promise<Array<unknown>>((resolve, reject) => {
    let pending = 0

    const next = async () => {
      const [id, index] = remaining.pop()!
      try {
        pending++
        documents[index] = await fetchDocument(id)
        pending--
      } catch (error) {
        return reject(error)
      }

      if (remaining.length > 0) {
        next()
      } else if (pending === 0) {
        resolve(documents)
      }
    }

    for (let i = 0; i < concurrency; i++) {
      next()
    }
  })
}
```

```ts
const fetchDocument = async (id: number, signal?: AbortSignal): Promise<unknown> => {
  const response = await fetch(`https://hacker-news.firebaseio.com/v0/item/${id}.json`, { signal: signal ?? null })
  return await response.json()
}

const fetchDocuments = async (ids: ReadonlyArray<number>, concurrency: number, signal?: AbortSignal): Promise<Array<unknown>> => {
  const controller = new AbortController()
  signal?.addEventListener("abort", () => controller.abort())

  const remaining = ids.slice().map((id, index) => [id, index] as const).reverse()
  const documents: Array<unknown> = []
  return new Promise<Array<unknown>>((resolve, reject) => {
    let pending = 0

    const next = async () => {
      const [id, index] = remaining.pop()!
      try {
        pending++
        documents[index] = await fetchDocument(id, controller.signal)
        pending--
      } catch (error) {
        controller.abort()
        return reject(error)
      }

      if (remaining.length > 0) {
        next()
      } else if (pending === 0) {
        resolve(documents)
      }
    }

    for (let i = 0; i < concurrency; i++) {
      next()
    }
  })
}
```

Interruption

Resilience

```ts
const wait = (ms: number) => new Promise<void>((resolve) => setTimeout(resolve, ms))

const withRetry = async <A>(fn: () => Promise<A>, delay: number = 1000, retries: number = 3): Promise<A> => {
  try {
    return await fn()
  } catch (error) {
    if (retries <= 0) {
      throw error
    }

    await wait(delay)
    return await withRetry(fn, delay, retries - 1)
  }
}

const fetchDocument = async (id: number, signal?: AbortSignal): Promise<unknown> =>
  withRetry(async () => {
    const response = await fetch(`https://hacker-news.firebaseio.com/v0/item/${id}.json`, {
      signal: signal ?? null
    })
    return await response.json()
  })
```

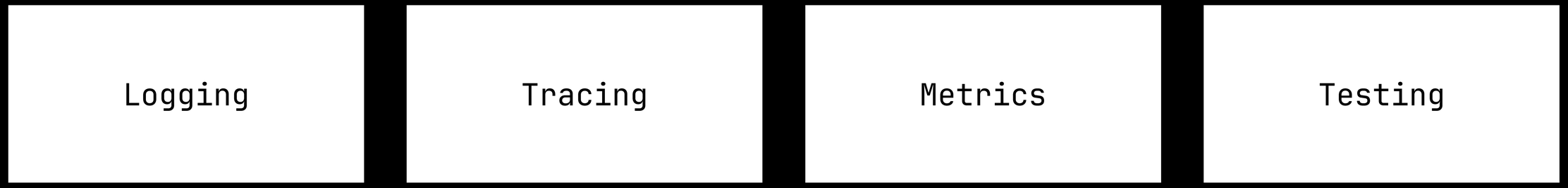
Interruption & Retries

```ts
const wait = (ms: number) => new Promise<void>((resolve) => setTimeout(resolve, ms))

const withRetry = async <A>(fn: () => Promise<A>, delay: number = 1000, retries: number = 3): Promise<A> => {
  try {
    return await fn()
  } catch (error) {
    if (retries <= 0) {
      throw error
    }

    await wait(delay)
    return await withRetry(fn, delay, retries - 1)
  }
}

const fetchDocument = async (id: number, signal?: AbortSignal): Promise<unknown> =>
  withRetry(async () => {
    const response = await fetch(`https://hacker-news.firebaseio.com/v0/item/${id}.json`, { signal: signal ?? null })
    return await response.json()
  })

const fetchDocuments = async (ids: ReadonlyArray<number>, concurrency: number, signal?: AbortSignal): Promise<Array<unknown>> => {
  const controller = new AbortController()
  signal?.addEventListener("abort", () => controller.abort())

  const remaining = ids.slice().map((id, index) => [id, index] as const).reverse()
  const documents: Array<unknown> = []
  return new Promise<Array<unknown>>((resolve, reject) => {
    let pending = 0

    const next = async () => {
      const [id, index] = remaining.pop()!
      try {
        pending++
        documents[index] = await fetchDocument(id, controller.signal)
        pending--
      } catch (error) {
        controller.abort()
        return reject(error)
      }

      if (remaining.length > 0) {
        next()
      } else if (pending === 0) {
        resolve(documents)
      }
    }

    for (let i = 0; i < concurrency; i++) {
      next()
    }
  })
}
```

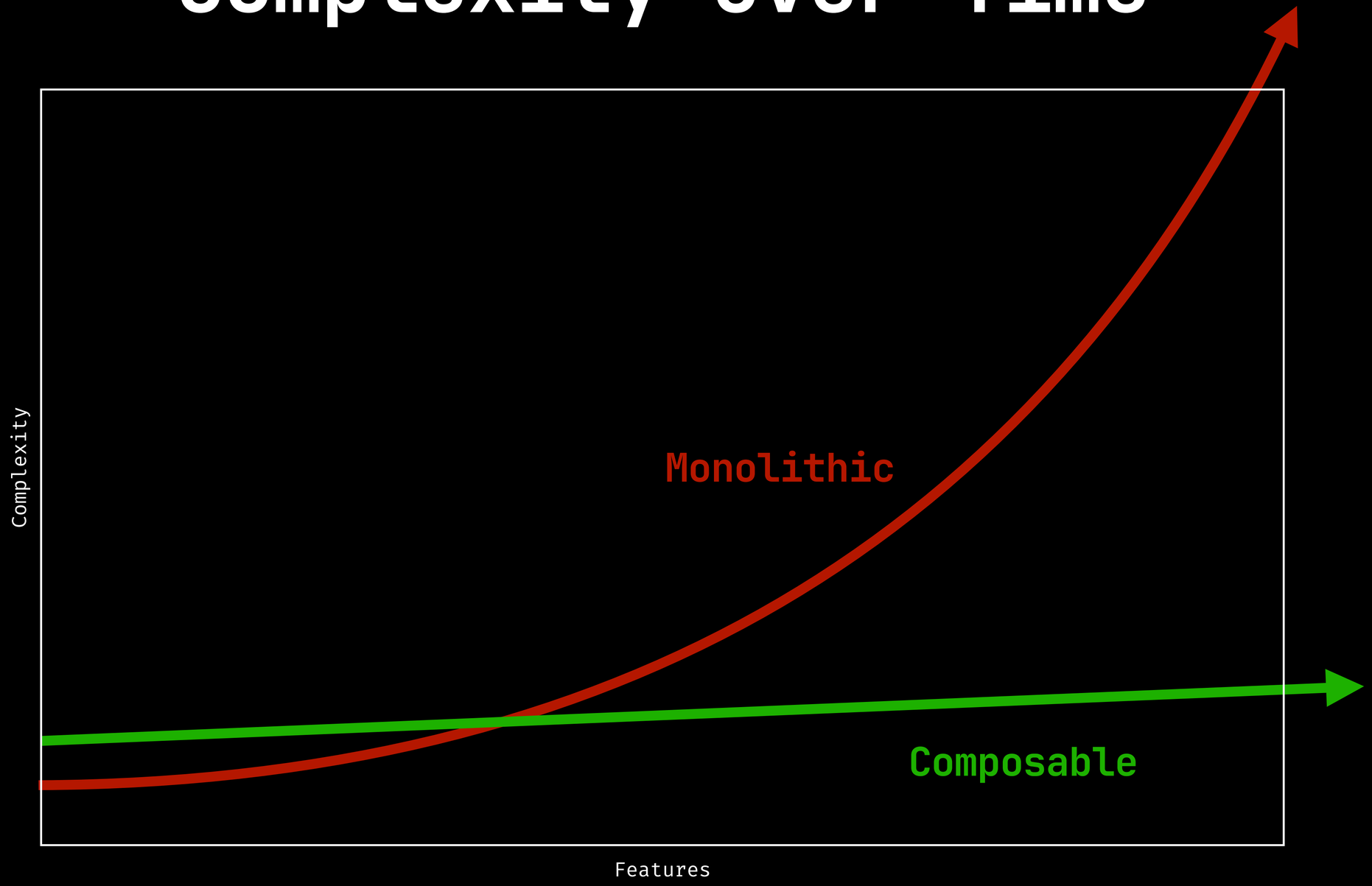
Opinionated Monolith

Leaky Abstraction

Incomplete Solution



Complexity Over Time



RUNTIME

CONCURRENCY / COMPOSABILITY / RESOURCE SAFETY / TYPE SAFETY / ERROR HANDLING / ASYNCHRONICITY / OBSERVABILITY

RUNTIME

CONCURRENCY / COMPOSABILITY / RESOURCE SAFETY / TYPE SAFETY / ERROR HANDLING / ASYNCHRONICITY / OBSERVABILITY

STANDARD LIBRARY

PATTERN MATCHING / TAGGED DATA TYPES / ENCODING / HASHING / ORDERING / STREAMING / DURATIONS / SCHEDULES / OPTION / EITHER / CHUNK / BIG DECIMALS / READONLY ARRAY / READONLY RECORD / HASH MAP / HASH SET / RED-BLACK-TREE ...

RUNTIME

CONCURRENCY / COMPOSABILITY / RESOURCE SAFETY / TYPE SAFETY / ERROR HANDLING / ASYNCHRONICITY / OBSERVABILITY

STANDARD LIBRARY

PATTERN MATCHING / TAGGED DATA TYPES / ENCODING / HASHING / ORDERING / STREAMING / DURATIONS / SCHEDULES / OPTION / EITHER / CHUNK / BIG DECIMALS / READONLY ARRAY / READONLY RECORD / HASH MAP / HASH SET / RED-BLACK-TREE ...

ECOSYSTEM

SCHEMA / CLI / CLUSTER / PRINTER / PLATFORM: FILESYSTEM, HTTP CLIENT & SERVER, RPC, WORKERS, COMMAND EXECUTOR ...

RUNTIME

CONCURRENCY / COMPOSABILITY / RESOURCE SAFETY / TYPE SAFE

PATTERN MATCHI
EITHER

ENG / DURATIONS / SCHEDULES / OPTION /
H MAP / HASH SET / RED-BLACK-TREE ...

RUNS EVERYWHERE AND
PAIRS WITH EVERYTHING

ECOSYSTEM

ER / PLATFORM: FILESYSTEM, HTTP CLIENT & SERVER, RPC, WORKERS, COMMAND EXECUTOR ...

RUNTIME

TYPED THROUGH AND THROUGH

PATTERN MATCHI
EITHE

RUNS
PAIRS WIT

ECOSYSTEM

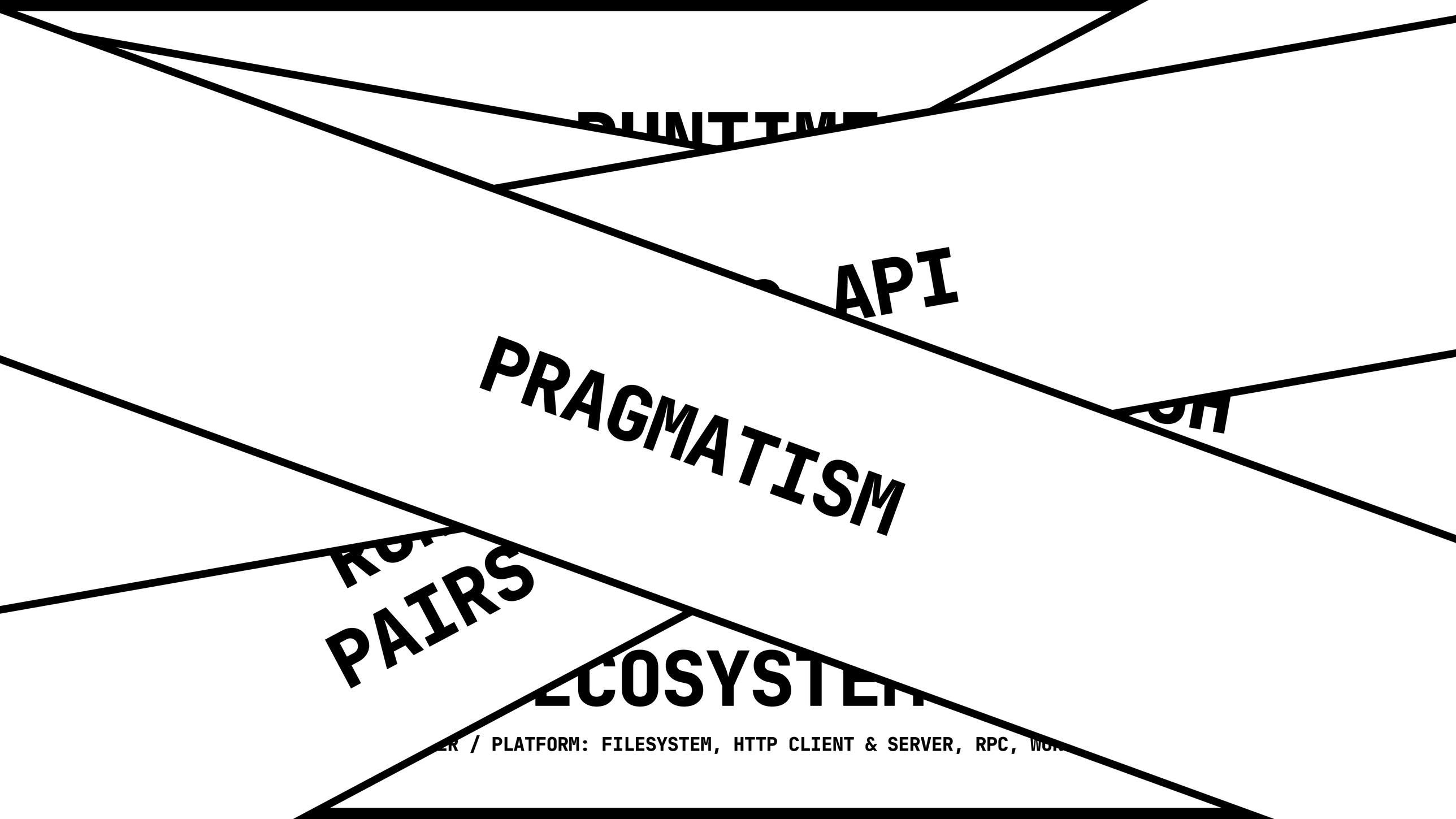
ER / PLATFORM: FILESYSTEM, HTTP CLIENT & SERVER, RPC, WORKERS, COMMAND EXECUTOR ...

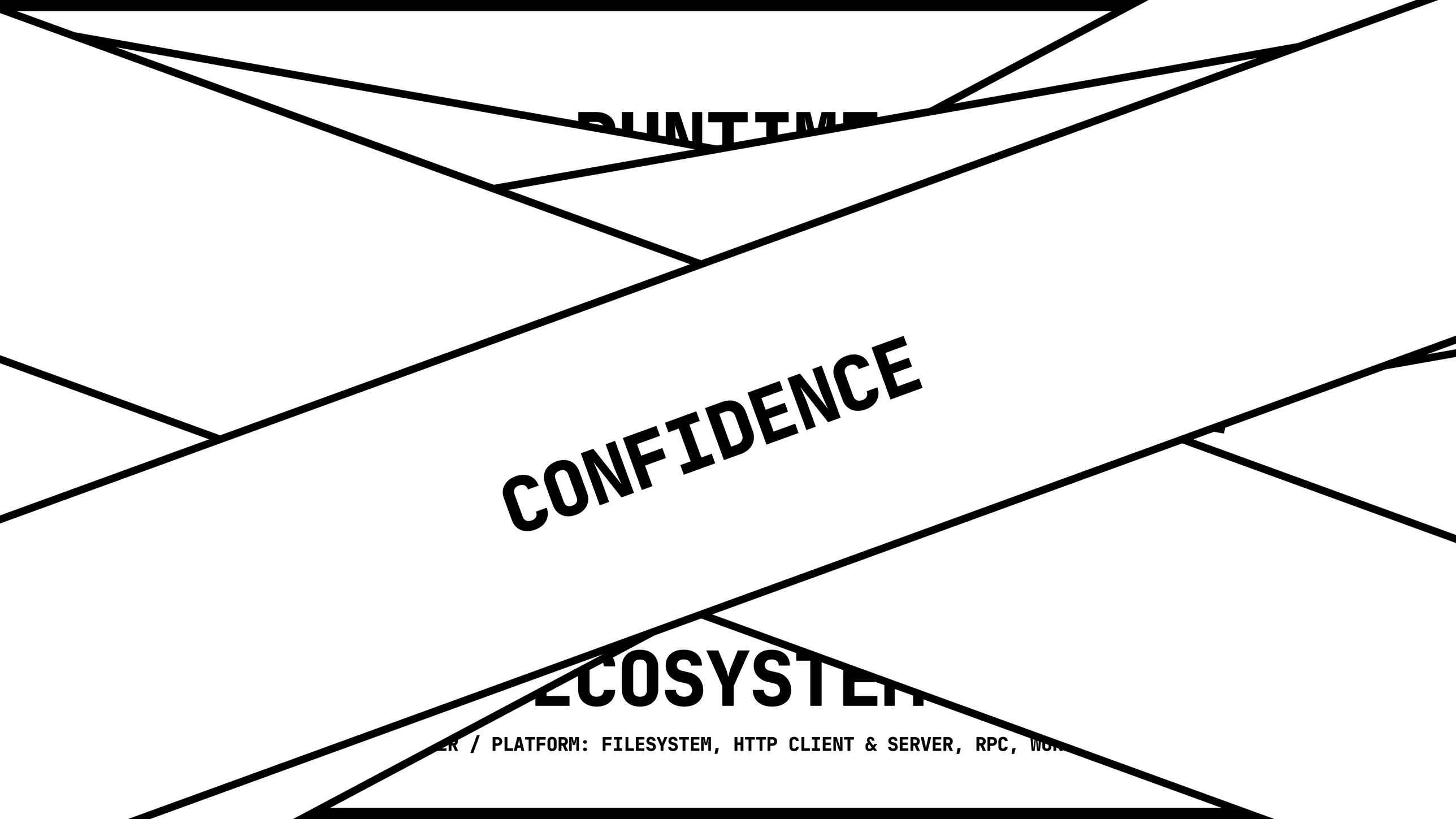
HOLISTIC API

PAIRS WIT

ECOSYSTEM

ER / PLATFORM: FILESYSTEM, HTTP CLIENT & SERVER, RPC, WORKERS, COMMAND EXECUTOR ...


API
PRAGMATISM
PAIRS
ECOSYSTEM
ER / PLATFORM: FILESYSTEM, HTTP CLIENT & SERVER, RPC, WO


RUNTIME
CONFIDENCE
ECOSYSTEM
ER / PLATFORM: FILESYSTEM, HTTP CLIENT & SERVER, RPC, WOR

```
Effect.Effect<R, E, A>
```

Requirements

Errors

Ausput

```
Effect.Effect<NumberGenerator, NumberTooLow, number>
```

Slides & Code

https://github.com/fubhy/effect-react-vienna-2023


Effect Days
Join Effect Days, the first-ever Effect conference
Thursday-Saturday, February 22-24, 2024
Urania, Uraniastraße 1, 1010, Vienna, Austria
Feb 22
Effect workshop
Feb 23
Conference day
Feb 24
Community day