Monitoring Windows

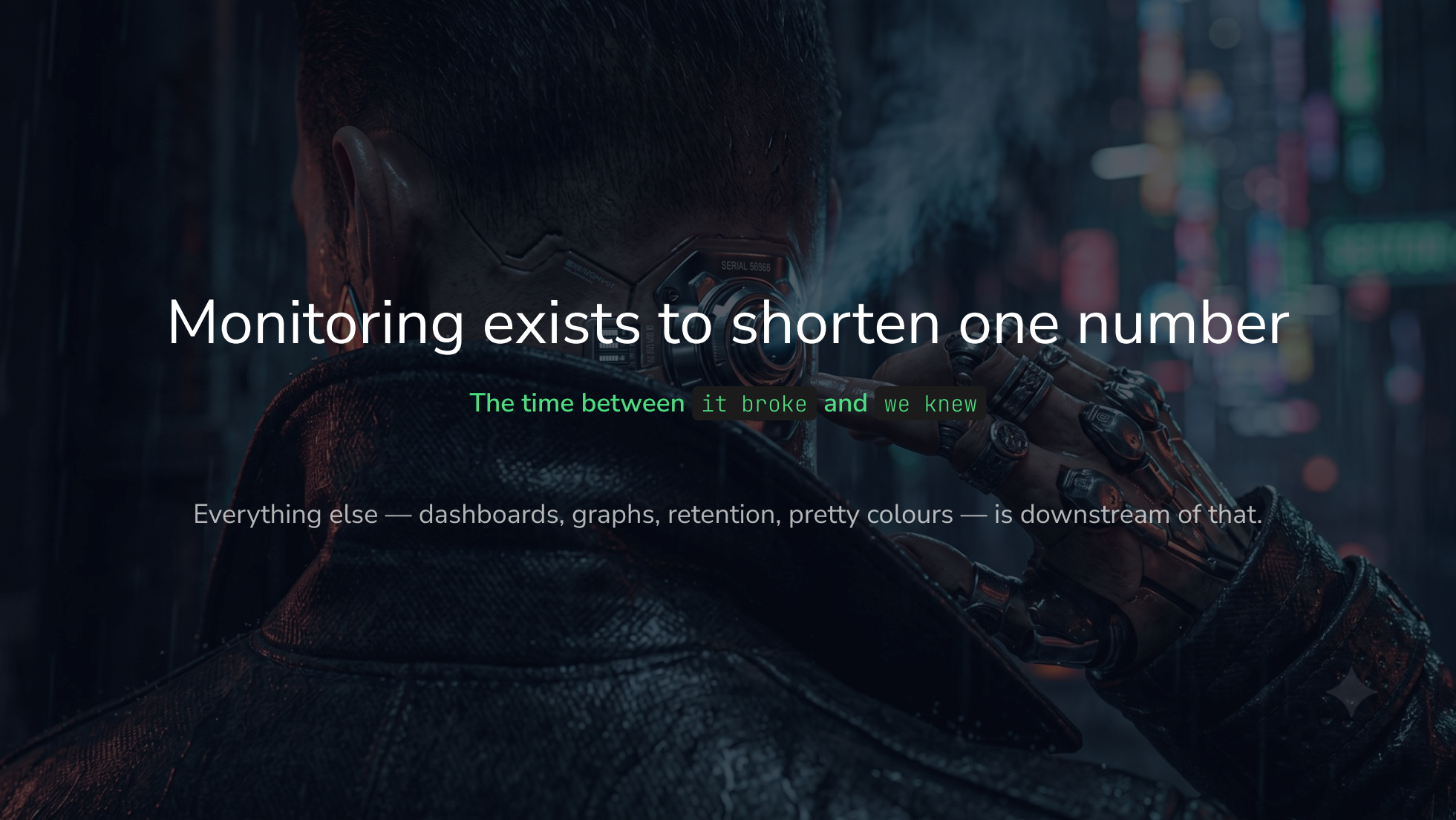
Not a tour of commands. A theory of what breaks, and how you find out.

Michael Medin · NSClient++

nsclient.org · github.com/mickem/nscp

1. Why we monitor

.and why most Windows monitoring doesn't



Monitoring exists to shorten one number

The time between `it broke` and `we knew`

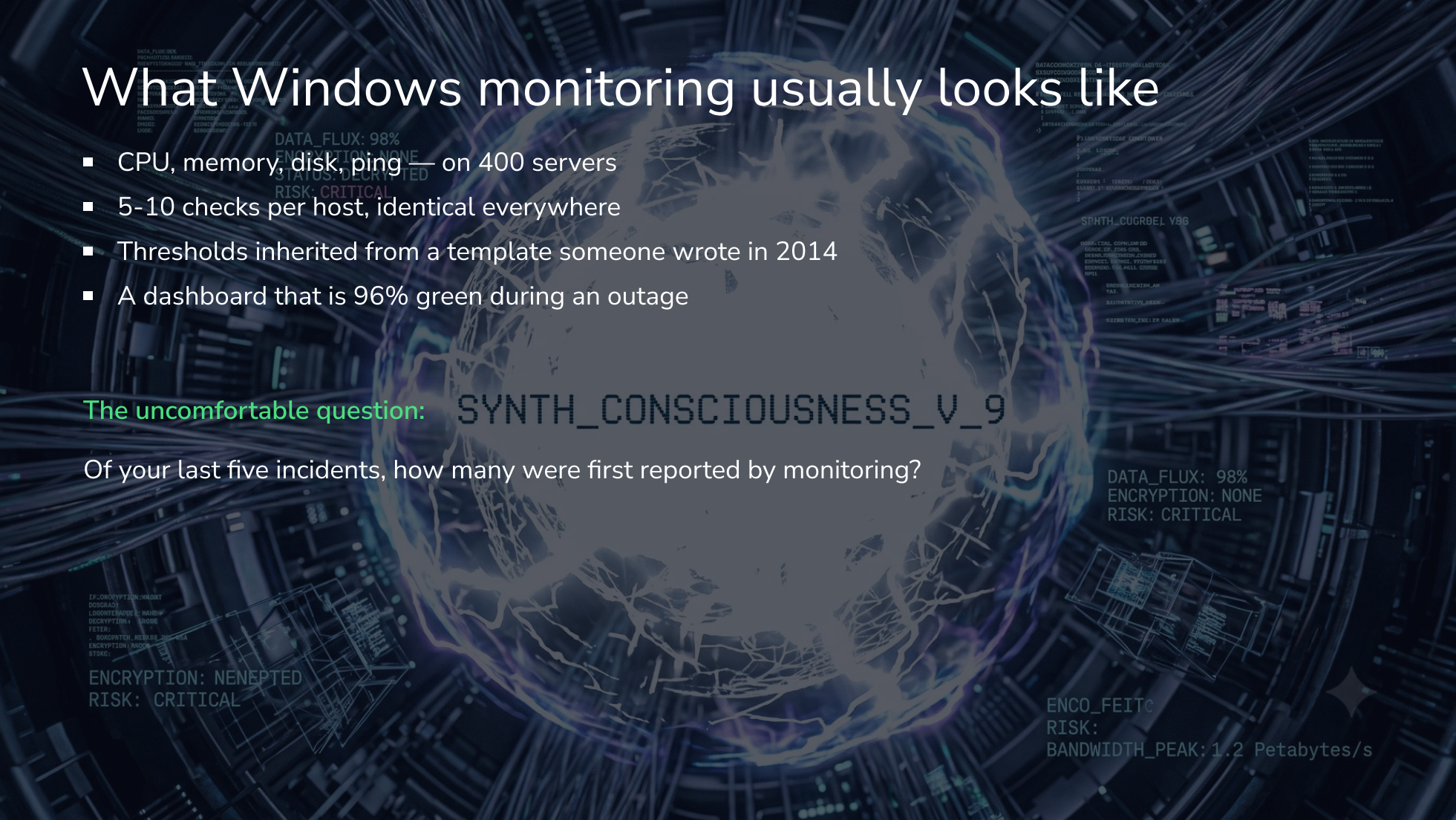
Everything else — dashboards, graphs, retention, pretty colours — is downstream of that.

What Windows monitoring usually looks like

- CPU, memory, disk, ping — on 400 servers
- 5-10 checks per host, identical everywhere
- Thresholds inherited from a template someone wrote in 2014
- A dashboard that is 96% green during an outage

The uncomfortable question:

Of your last five incidents, how many were first reported by monitoring?



Cause monitoring vs symptom monitoring

Cause

"A thing inside is unusual"

- CPU at 95%
- Memory at 88%
- Queue depth 4000
- Disk I/O latency up

Answers: *why*

Symptom

"Something someone cares about is broken"

…pen

Alert on symptoms.
Graph the causes.

Cause monitoring vs symptom monitoring

Cause

Windows hands you these

- Thousands of counters, out of the box
- Same check works on every server
- One template, four hundred hosts
- Nobody has to know what the box is *for*

Cost: *nothing*

Symptom

You have to build these

- No counter says "orders aren't being written"
- Different for every service
- Someone has to write down what "working" means
- Requires knowing what the box is *for*

Cost: *a conversation, then a check*

A useful test for any check

If this fires at 03:00,
is there something a human should do?

If no → it's a graph, not an alert.

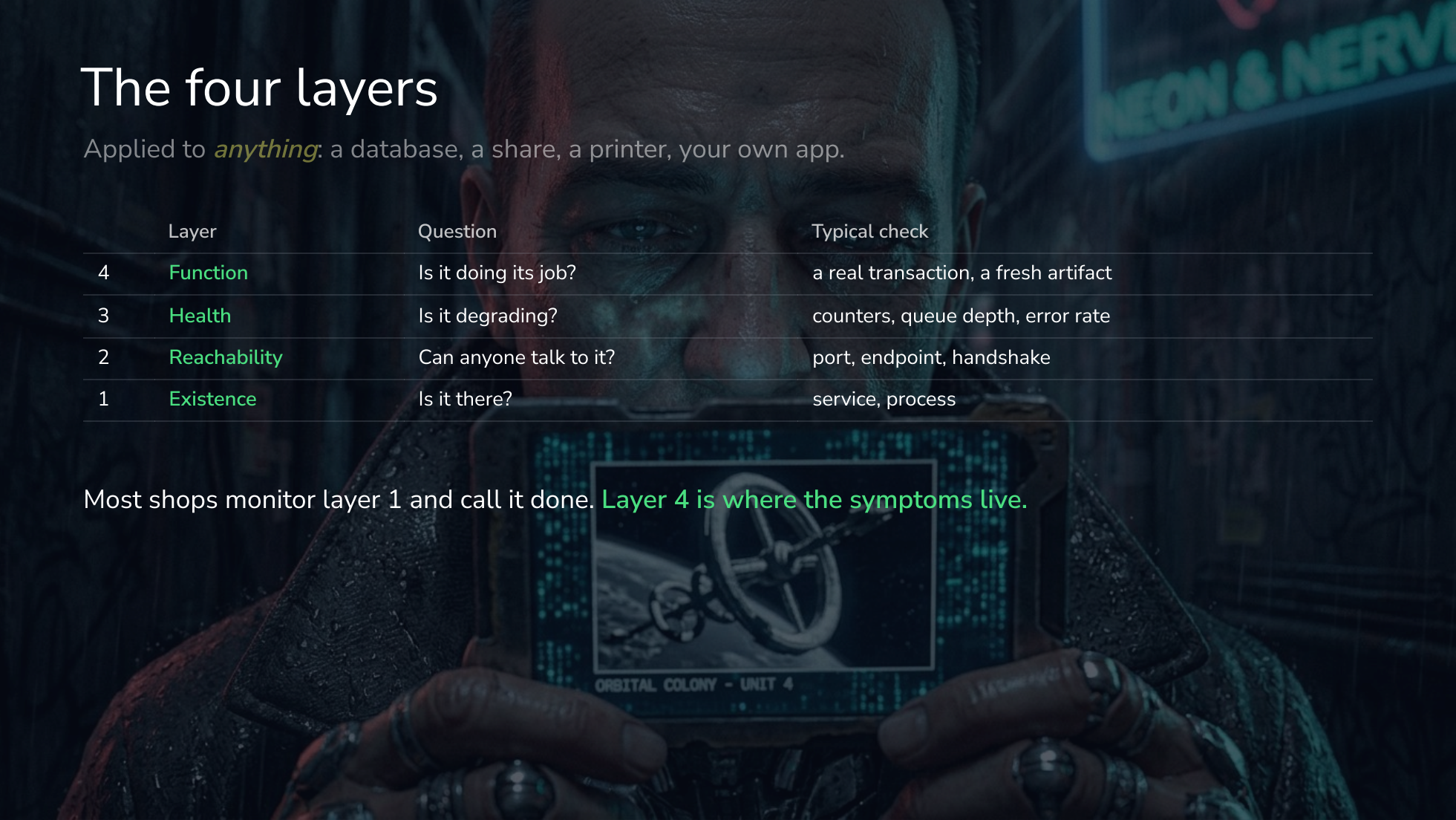
If yes → *where's the runbook?*

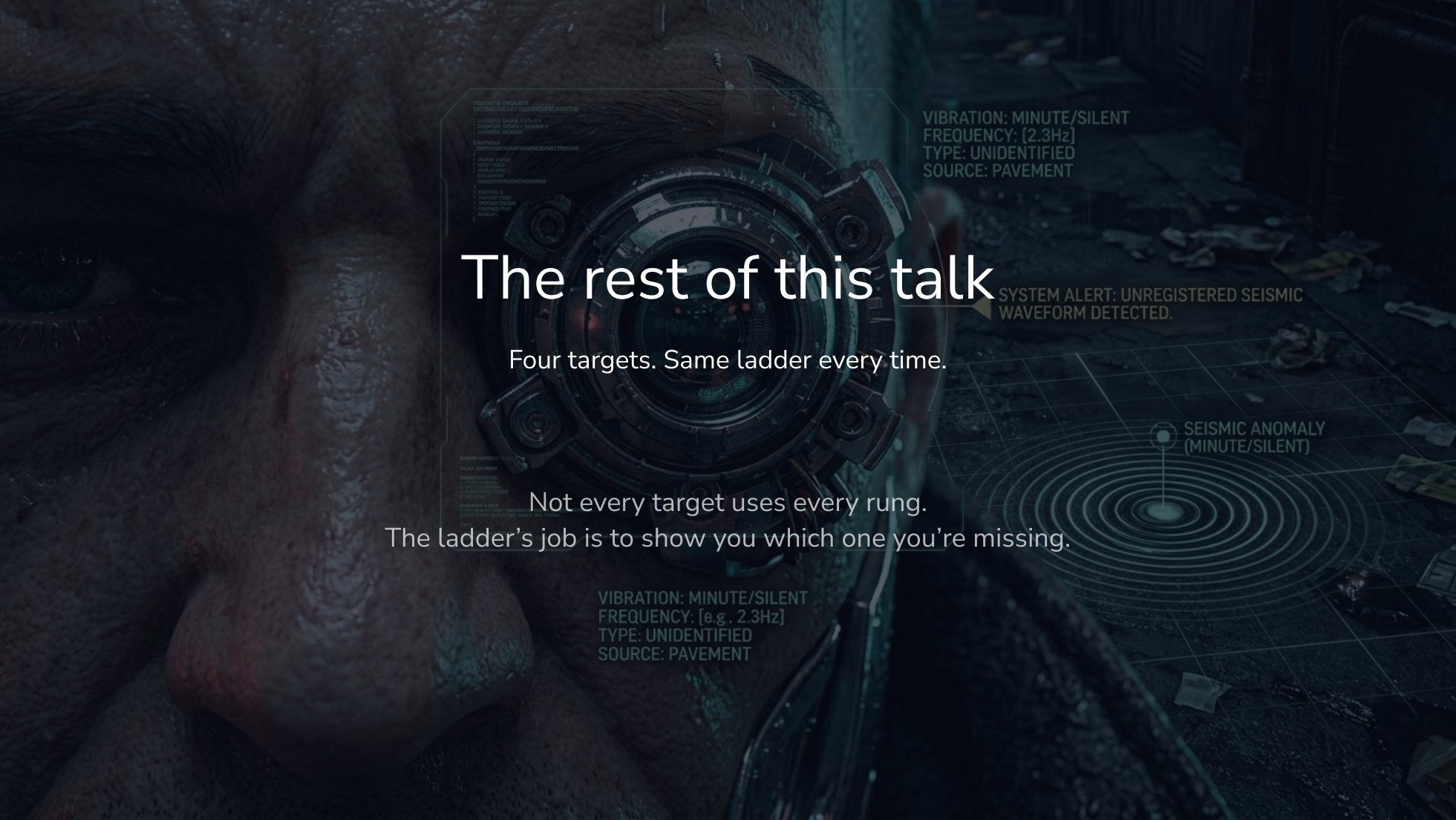
The four layers

Applied to *anything*: a database, a share, a printer, your own app.

	Layer	Question	Typical check
4	Function	Is it doing its job?	a real transaction, a fresh artifact
3	Health	Is it degrading?	counters, queue depth, error rate
2	Reachability	Can anyone talk to it?	port, endpoint, handshake
1	Existence	Is it there?	service, process

Most shops monitor layer 1 and call it done. Layer 4 is where the symptoms live.





The rest of this talk

Four targets. Same ladder every time.

Not every target uses every rung.
The ladder's job is to show you which one you're missing.

2. How you get the data

agentless vs agent · pull vs push · and who you are

Agentless vs agent

	Agentless WMI / SNMP / WinRM from the server	Agent something local on the box
Deployment	nothing to install	install, upgrade, maintain — forever
Credentials	domain account, on the wire, stored centrally	none needed for local queries
Firewall	Can be a challange	one port, or outbound-only
API reach	whatever WMI exposes	full Win32, PDH, event log, filesystem
Cost per check	remote WMI is slow and expensive on the target	local, milliseconds
Local scripts	no	yes
Event-driven	Rare and painfull	possible
"Host down"	indistinguishable from "WMI is broken"	agent silence is itself a signal

Agentless wins when you can't install software, when the fleet is tiny, or when it's someone else's box.

It's not wrong — it's a different trade-off.

Pull vs push

Pull — NRPE, WMI, SNMP

The server asks. The agent answers.

- Schedule lives centrally ✅
- Config lives centrally ✅
- Needs inbound access to every host ❌
- Silence is immediately visible ✅
- Poll interval = floor on detection ❌

Push — NSCA, NRDP

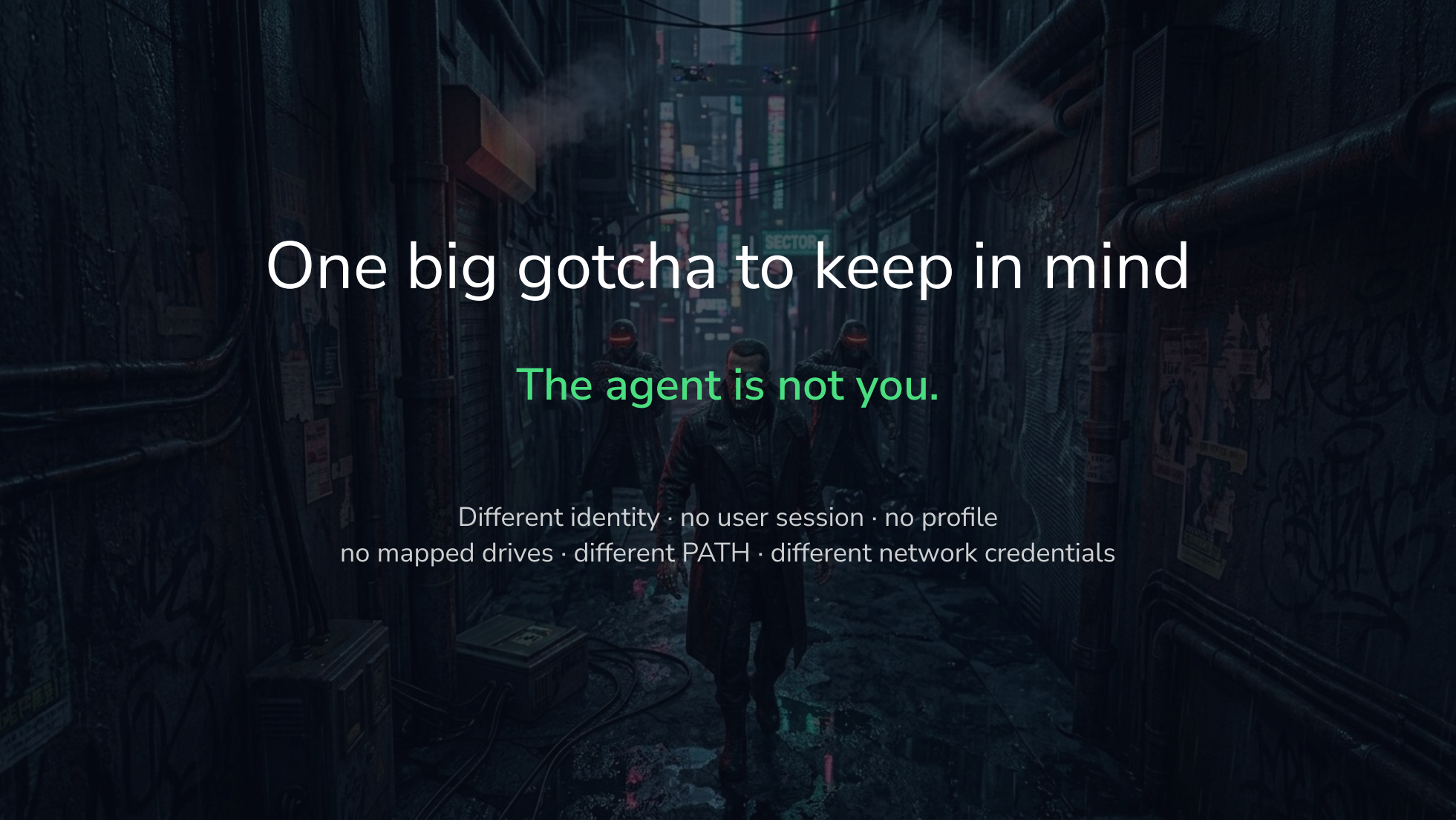
The agent decides and sends.

- Outbound only — NAT, DMZ, cloud ✅
- Scales: no central scheduler ✅
- Can be event-driven, not timed ✅
- Config is distributed ❌
- Silence needs freshness checks ❌

With a 5-minute poll: *your best possible detection time is 5 minutes.*

Push on event makes it seconds — for the handful of things where that's worth it.

Fleet server: removes the pain of distributed config



One big gotcha to keep in mind

The agent is not you.

Different identity · no user session · no profile
no mapped drives · different PATH · different network credentials

Enough theory

Let's monitor something.

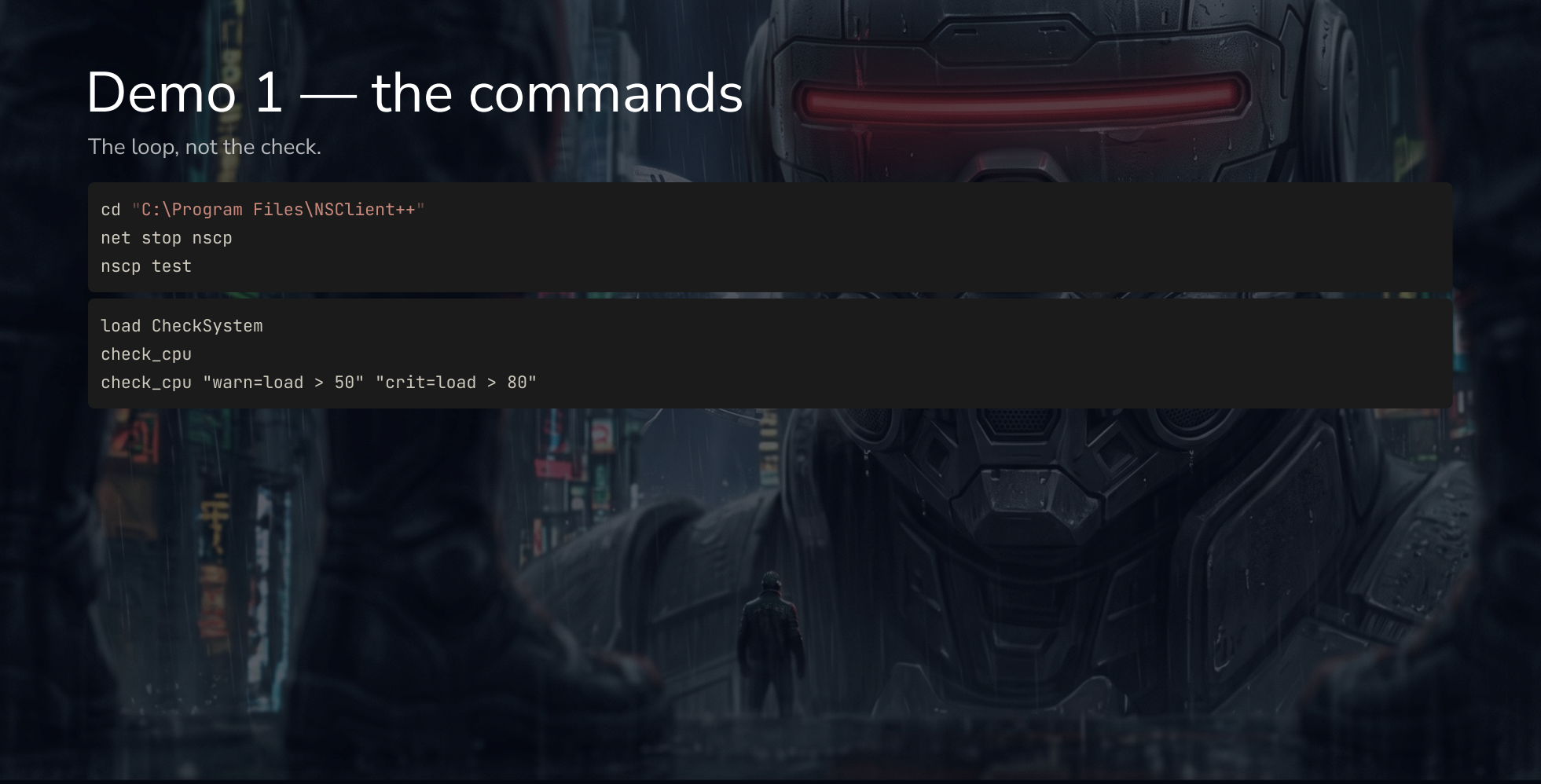
Terminal tab · `nscp test`

Demo 1 — the commands

The loop, not the check.

```
cd "C:\Program Files\NSClient++"
net stop nscp
nscp test
```

```
load CheckSystem
check_cpu
check_cpu "warn=load > 50" "crit=load > 80"
```

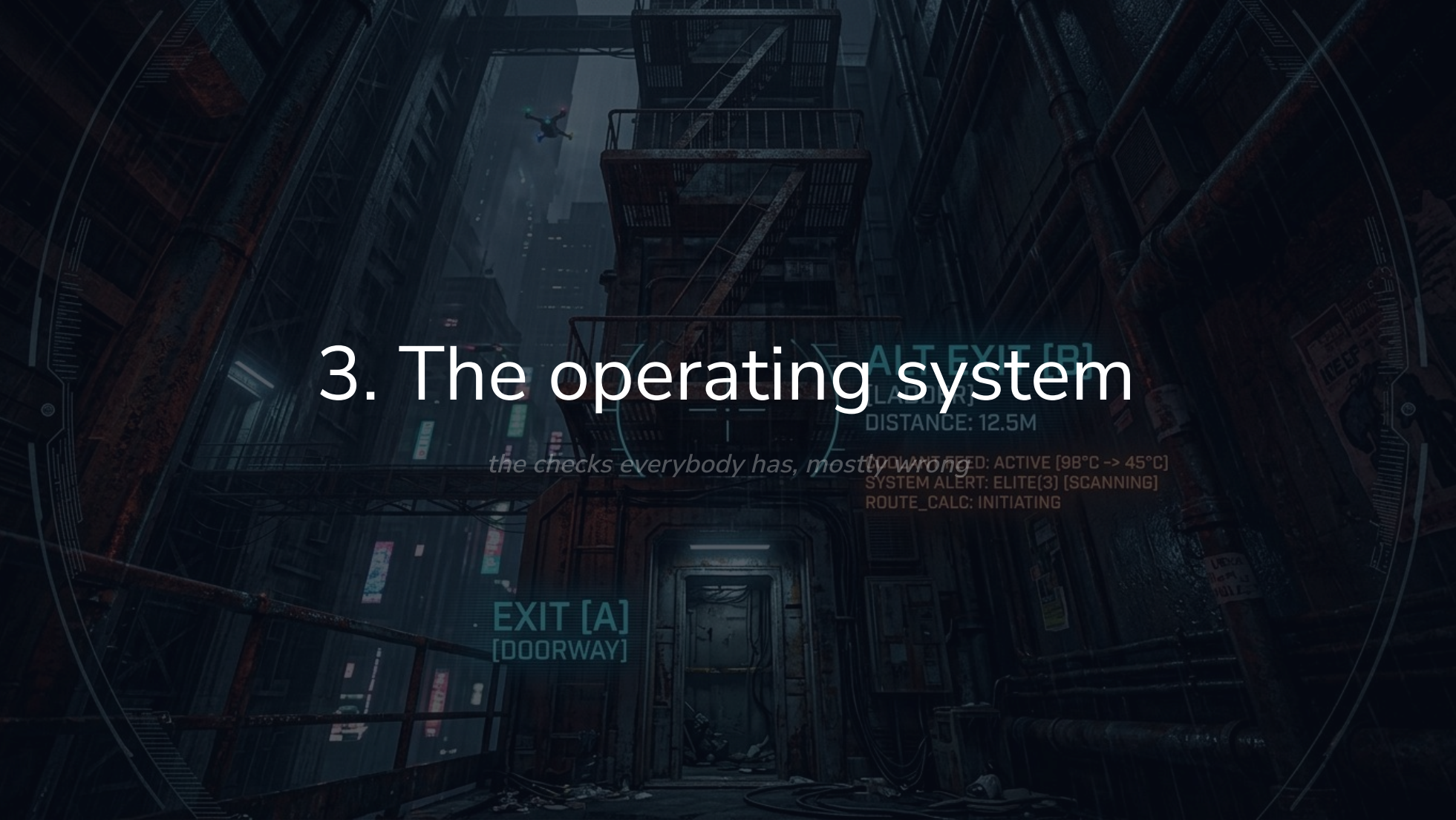


One grammar, every check

`check_service`	`"filter=start_type = 'auto'"`	`"crit=state != 'started'"`	`"detail-syntax=${name} is ${state}"`
source	filter	threshold	syntax
where from	what's interesting	what's bad	how to say it

- source — the check command: files, counters, WMI, HTTP, event log…
- filter — narrow the set. *Start with* `show-all` *, then narrow.*
- threshold — a filter that means "bad". Same language.
- syntax — what the human reads at 03:00, and what gets graphed

Learn it once. Every check you've never seen is already 80% familiar.



3. The operating system

the checks everybody has, mostly wrong

What actually takes a Windows box down

Ranked by how often it's the real cause:

1. A volume filled up — logs, WinSxS, update cache, a runaway trace file
2. A service didn't come back after a patch reboot
3. A process leaked memory until the box started paging itself to death
4. The clock drifted — Kerberos fails, auth breaks, nothing looks wrong
5. A pending reboot sat unnoticed for six weeks
6. CPU was high

Note where CPU is.

Note that everyone alerts on it.

Thresholds are the hard part

Static percentage — `disk > 90%`

- Fine on uniform, slow-growing volumes
- On a 4 TB volume, 10% free is 400 GB
- On a 40 GB volume, 10% is four minutes

Absolute free — `free < 5 GB`

- Correct for big volumes
- Wrong for small ones
- Needs per-host tuning, which nobody does

Rate of change → time to exhaustion

This volume runs out in 4 hours is actionable.

This volume is 91% full is not.

```
check_drivesize "warn=full_in < 2d" "crit=full_in < 4h"
```

Existence: Used to be simple

Layer 1 — and for a *box*, layers 1 and 3 are the whole ladder. Nothing connects to an operating system, and "doing its job" is whatever runs on top of it.

ch

-
-
-
-

Fi
A

This is why we never inherited the monitoring-plugins -w and -c.

A number can't say "except that one".

Demo

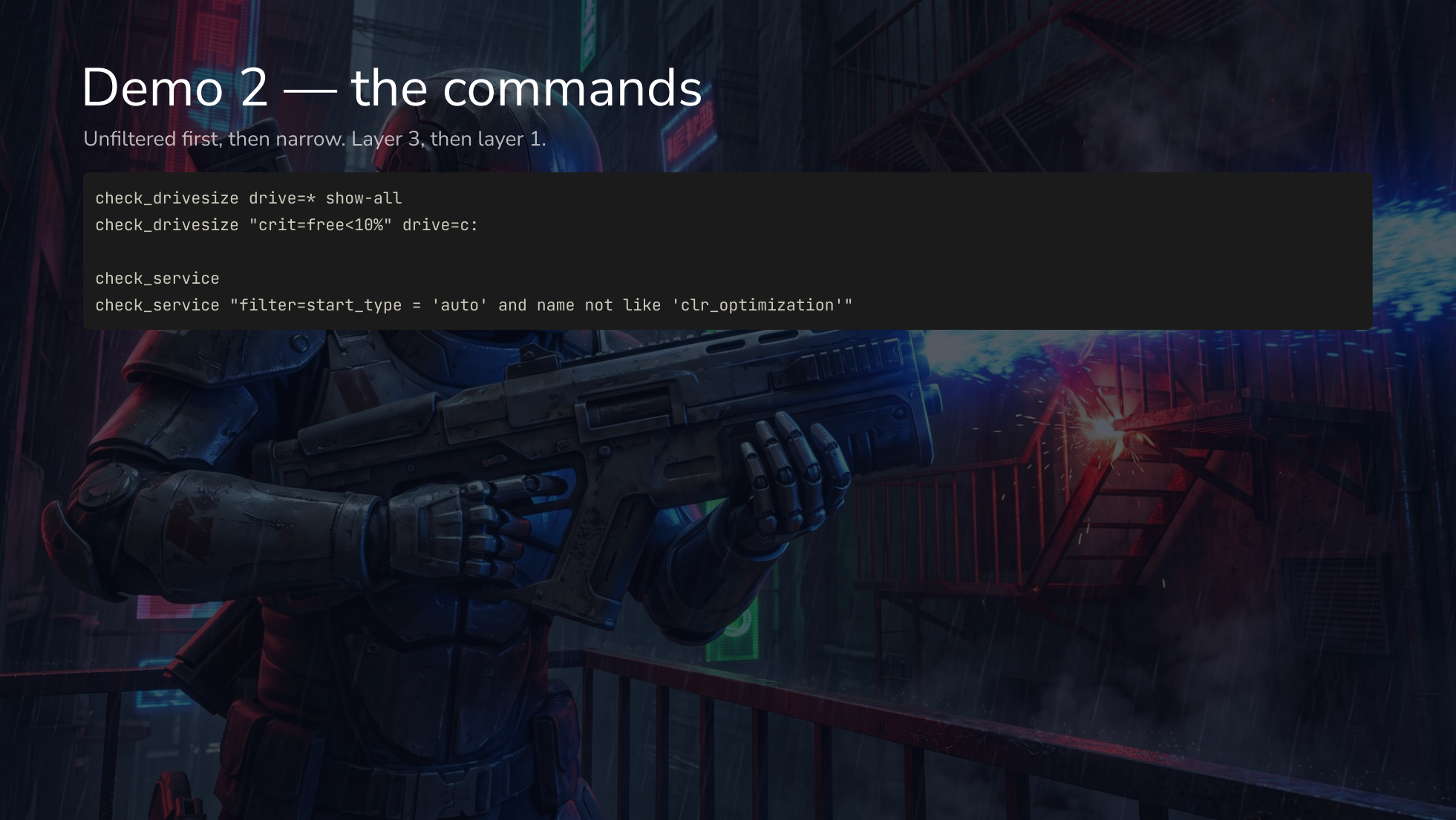
Show everything → then narrow

Demo 2 — the commands

Unfiltered first, then narrow. Layer 3, then layer 1.

```
check_drivesize drive=* show-all
check_drivesize "crit=free<10%" drive=c:

check_service
check_service "filter=start_type = 'auto' and name not like 'clr_optimization'"
```



The rest of the module

Photograph this slide rather than watching me type it.

Classic

```
check_cpu          check_memory
check_uptime       check_pagefile
check_network      check_process
```

Newer, and useful

```
check_os_version           check_os_updates
check_pending_reboot       check_patch_age
check_disk_health          check_disk_io
check_installed_software   check_shadowcopy
check_process_history      check_tasksched
```

Security posture (new module)

```
check_defender         check_firewall
check_bitlocker        check_certificate
```

Dont forget:

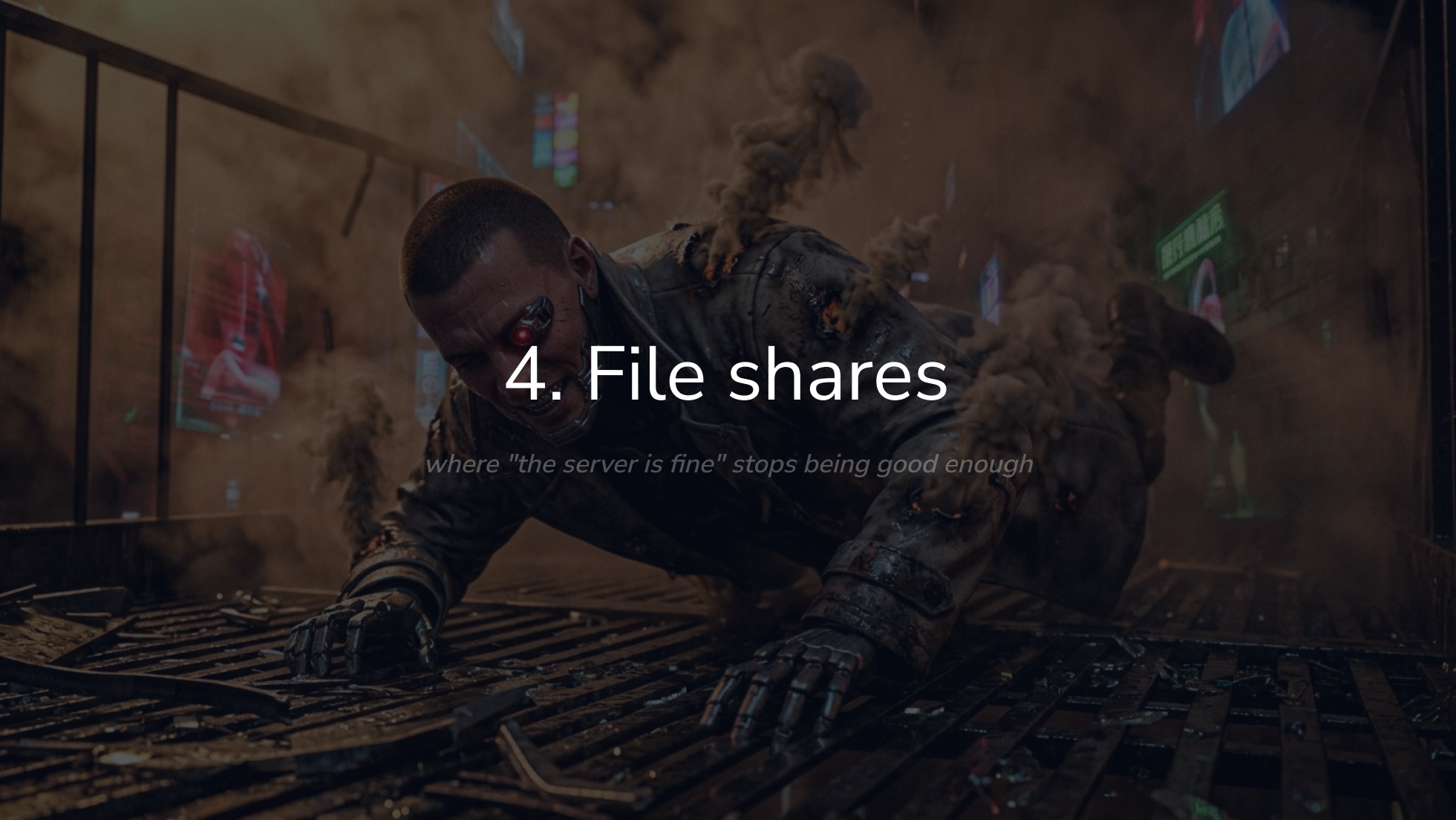
`check_ntp_offset` — Clock drift are hard to catch.

`check_w32time` what the Windows Time service thinks it's doing.

`check_pending_reboot` — and how long it's been pending.

`check_patch_age` — how stale is this box.

`check_nscp_update` — how stale is the agent.



4. File shares

where "the server is fine" stops being good enough

Why shares are worth talking about

- They're infrastructure other things depend on — a share failing is never just a share failing
- Humans notice immediately, and they notice before your monitoring does
- The failure modes are almost all invisible from the server

The server says: *the service is running, the share is published, the disk has space.*

The user says: *I can't save my file.*

Both are true at the same time, routinely.

Provider vs experience

The provider

from the server that exports it

```
check_service service=LanmanServer
check_share share=finance
check_drivesize drive=D:
check_files path=D:\shares\finance
```

- ✅ Cheap, always available, no extra host
- ✅ Tells you *why* when it breaks
- ❌ Proves nothing about usability
- ❌ Blind to ACLs, quotas, DFS, sessions

The experience

from a client that consumes it

```
check_uncpath path=\\srv\finance
check_files path=\\srv\finance\drop
check_disk_write file=\\srv\finance\probe.dat
```

- ✅ Proves the actual thing users do
- ✅ Catches ACL / quota / DFS / resolution
- ❌ Needs a probe host and an identity
- ❌ Network blips become share alerts

The experience check is the symptom.

The provider checks are the causes.

And it's the four layers again: the provider proves 1 and 3. Only a client can prove 2 and 4.

Demo

Two commands. One of them fails.

Demo 3 — the commands

Both are layer 2 — the rung the provider cannot see.

```
check_drivesize drive=f:                  # fails: no session, so no mapped drive
check_uncpath   path=\127.0.0.1\finance   # works: a UNC path is not session state
```

And when the share does not grant the machine account:

```
check_uncpath path=\127.0.0.1\backups     # fails: machine account has no access
check_uncpath path=\127.0.0.1\backups user=.\svc_monitor password=...
```

Mapped drive: fails
UNC path: works

Because mapped drives are per-session,
and the agent doesn't have a session.

...and the UNC path only works because LocalSystem
presents the machine account, which this share happens to trust.

When it doesn't: `check_uncpath user= password=` —
the identity problem, solved without a script.


5. SQL Server

"Is SQL down?" is the wrong question

It is almost never down. It degrades, and degradation is invisible until it isn't.

What actually goes wrong	Visible in "is the service running"?
Backups silently stopped working	❌
Transaction log filled → database refuses writes	❌
tempdb filled	❌
One long transaction blocking forty sessions	❌
Log-volume disk latency went from 2 ms to 200 ms	❌
Availability Group replica falling behind	❌
The service actually crashed	✅

Six of seven are invisible at layer 1. And layer 1 is what most shops monitor.

Where do you get SQL truth from?

Source	Sees	Costs you
Service state	a process exists	nothing
TCP port `1433`	a listener accepting	nothing
Performance counters	engine-wide health	nothing
WMI	configuration, some state	WMI being healthy
T-SQL / DMVs	*everything* — blocking, log space, replica lag	a login + query load
Backup files on disk	the truth about backups	file access
Event log	crashes, corruption, login failures	nothing

- Counters are the sweet spot for health — no auth, no query load, whole-engine coverage
- DMVs are the only real truth — and as of this year the agent speaks T-SQL natively, so the price has dropped to a login
- Backup files cost nothing and catch the worst failure

New this year: the agent speaks T-SQL

`check_mssql`	can I connect — version, patch level, edition, uptime
`check_mssql_databases`	every database ONLINE — recovery model, log usage
`check_mssql_backup`	age of the last full / diff / log backup, per database
`check_mssql_jobs`	SQL Server Agent job status
`check_mssql_query`	your own T-SQL, thresholds on the result

```
[/settings/mssql]
; empty user + password = Windows auth, as the agent's service account
; server, driver, timeouts set once — every check inherits
```

No password to store. Grant the service account a read-only login instead.

SQL’s symptoms, SQL’s causes

Symptoms

Someone is already having a bad day

- Orders aren’t being written
- The application is timing out
- Logins are being refused
- Last night’s backup didn’t run

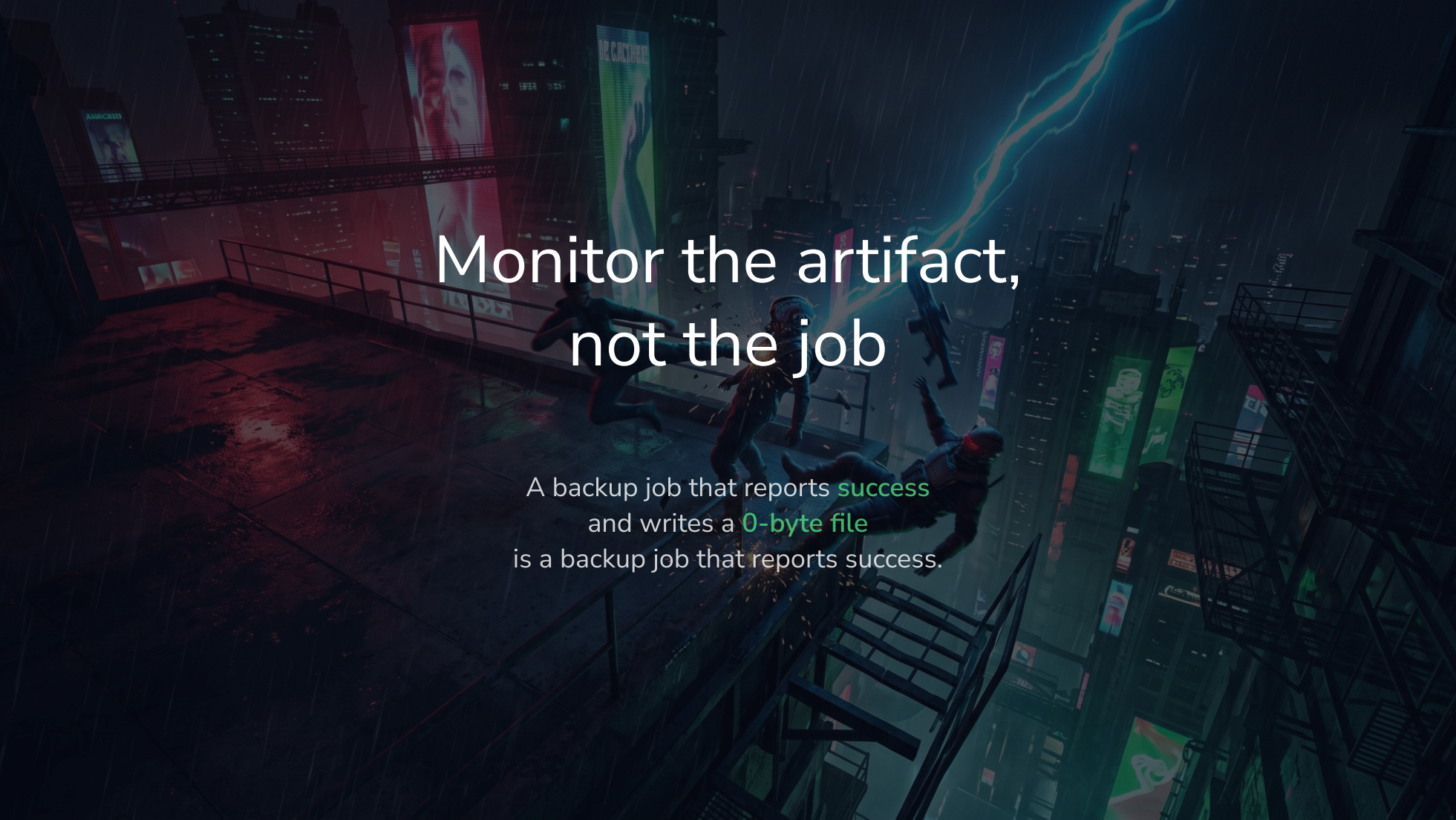
Causes — the ones that buy you time

Hours of warning, if anyone is watching

- Page life expectancy falling
- Log space consumed climbing
- Blocking duration creeping
- Disk queue length growing

Same rule as the start: alert on the left, graph the right. *What’s new here — the right-hand column is early enough to act on.*

The mistake: paging on one of those causes with a static threshold. PLE below 300 is a famous rule of thumb that is wrong on most modern hardware — and paging on it teaches people that SQL alerts are noise.



Monitor the artifact, not the job

A backup job that reports success
and writes a 0-byte file
is a backup job that reports success.

Demo

Climbing all four rungs

Demo 4 — the commands

Each layer twice: the generic way, then the built-in.

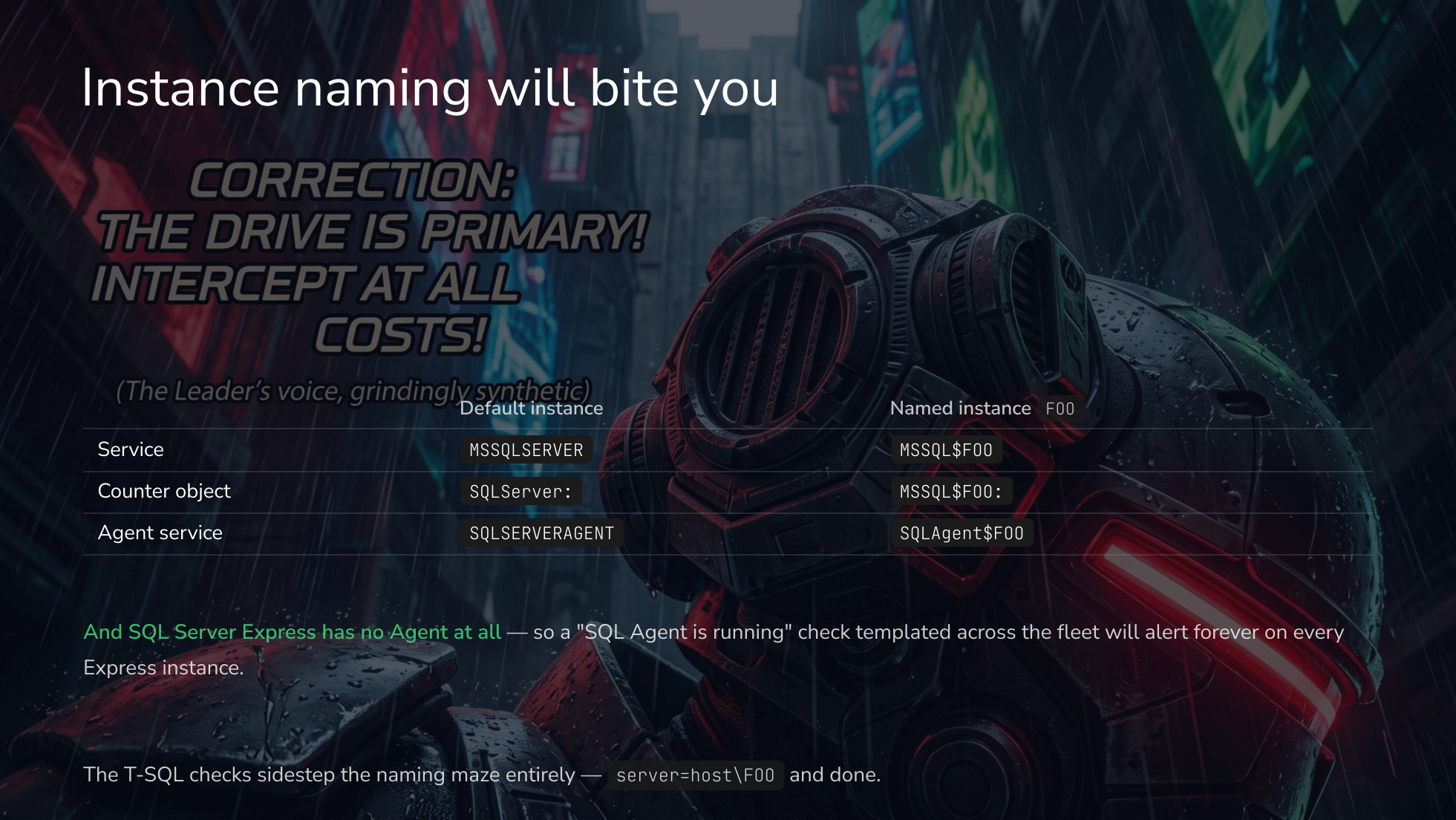
```
# 1  is it there?
check_service "filter=name like 'MSSQL'" "crit=state != 'started'"

# 2  is it reachable?
check_tcp host=127.0.0.1 port=1433
check_mssql

# 3  is it degrading?
check_pdh "counter:ple=\SQLServer:Buffer Manager\Page life expectancy" "crit=value < 300"
check_mssql_databases "detail-syntax=${name}: ${state} log used ${log_used_pct}%" show-all

# 4  is it doing its job?
check_files path=C:\SQLBackup pattern=*.bak "crit=written < -26h"
check_mssql_backup "critical=full_age = -1 or full_age > 26h"
```

Instance naming will bite you

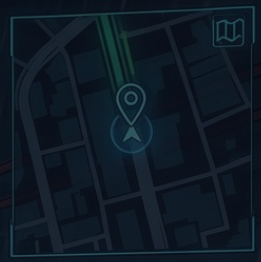


	Default instance	Named instance `FOO`
Service	`MSSQLSERVER`	`MSSQL$FOO`
Counter object	`SQLServer:`	`MSSQL$FOO:`
Agent service	`SQLSERVERAGENT`	`SQLAgent$FOO`

And SQL Server Express has no Agent at all — so a "SQL Agent is running" check templated across the fleet will alert forever on every Express instance.

The T-SQL checks sidestep the naming maze entirely — `server=host\FOO` and done.

What I’d actually monitor on a SQL box

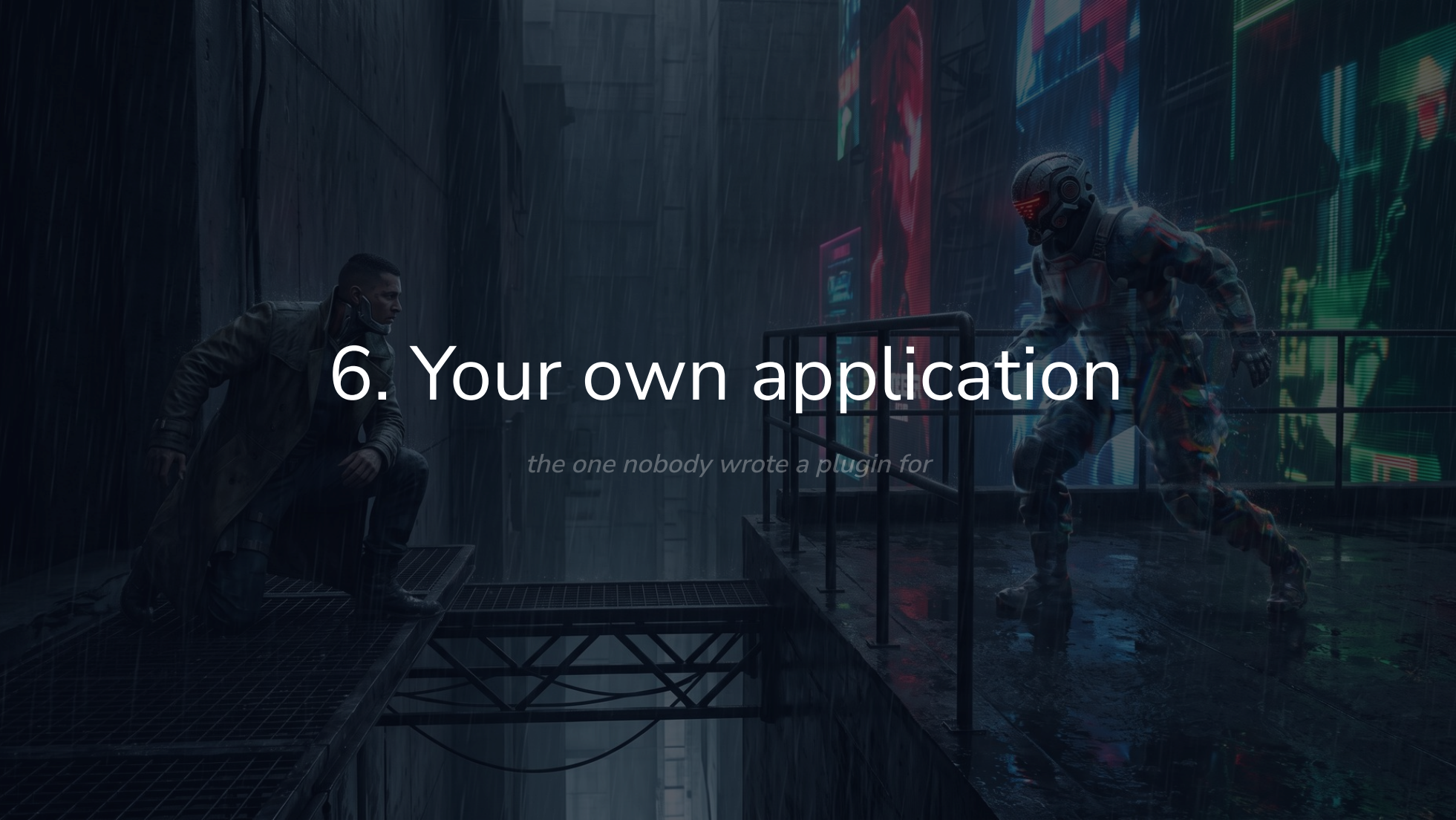


Page

- Backup artifact older than expected
  `check_files path=C:\SQLBackup pattern=*.bak "crit=written < -26h"`
- Database not ONLINE / read-only
  `check_mssql_databases` — the defaults already do this
- Log space > 85% *and rising*
  `check_mssql_databases "warn=log_used_pct > 85"`
- Service down, port not listening
  `check_service "filter=name like 'MSSQL'" "crit=state != 'started'"`
  `check_tcp host=127.0.0.1 port=1433`
- AG replica lag beyond RPO
  `check_mssql_query "query=…" "crit=lag_s > 60"`

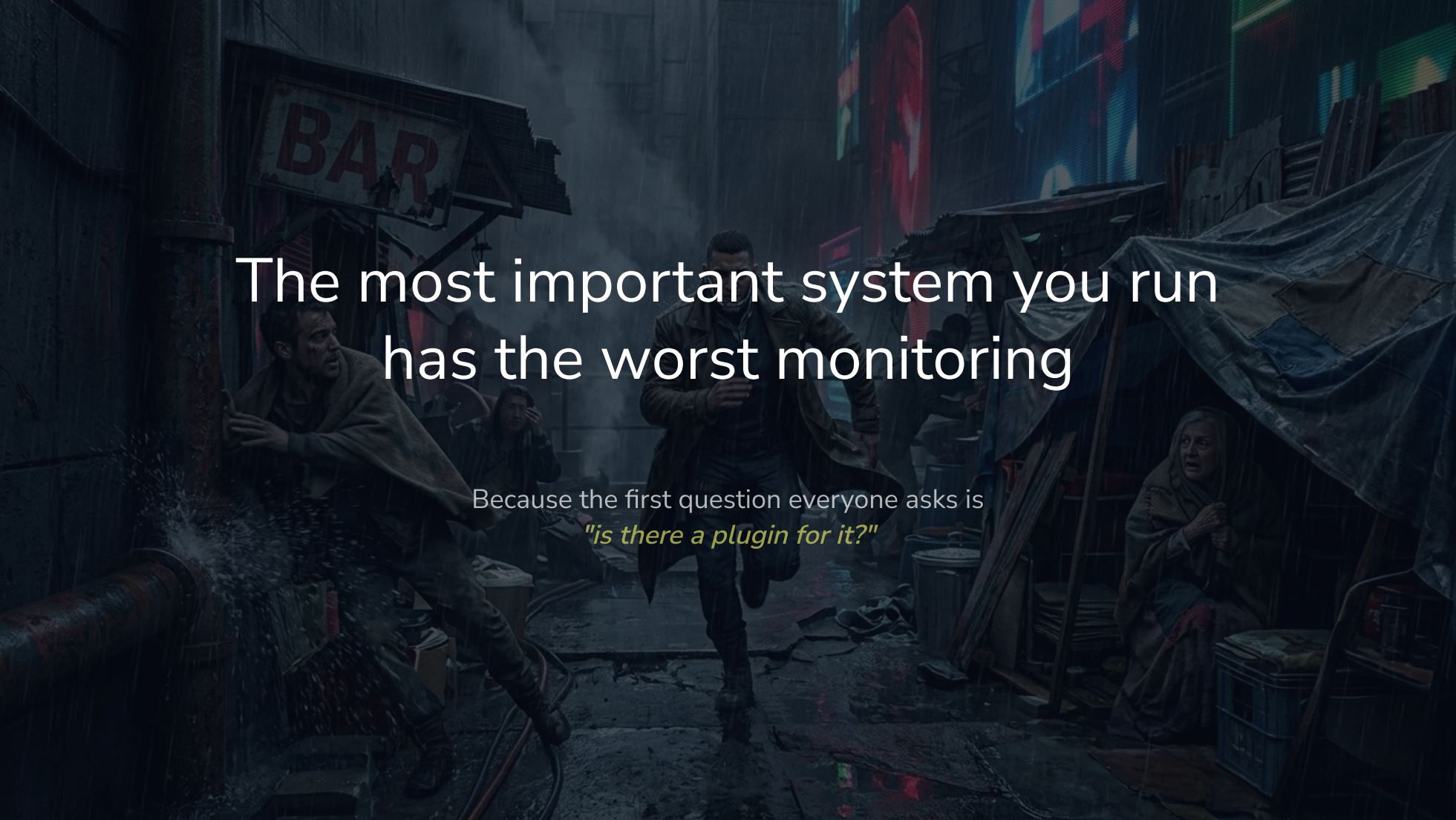
Graph, don’t page

- Page life expectancy
  `check_pdh "counter:ple=\SQLServer:Buffer Manager\Page life expectancy"`
- Batch requests/sec, user connections
  `check_pdh "counter:b=\SQLServer:SQL Statistics\Batch Requests/sec"`
- Disk queue length, I/O per volume
  `check_disk_io "warn=queue_length > 4"`
- Lock waits, blocking duration
  `check_pdh "counter:lw=\SQLServer:Locks(_Total)\Lock Waits/sec"`
- tempdb volume growth
  `check_drivesize drive=t: show-all`



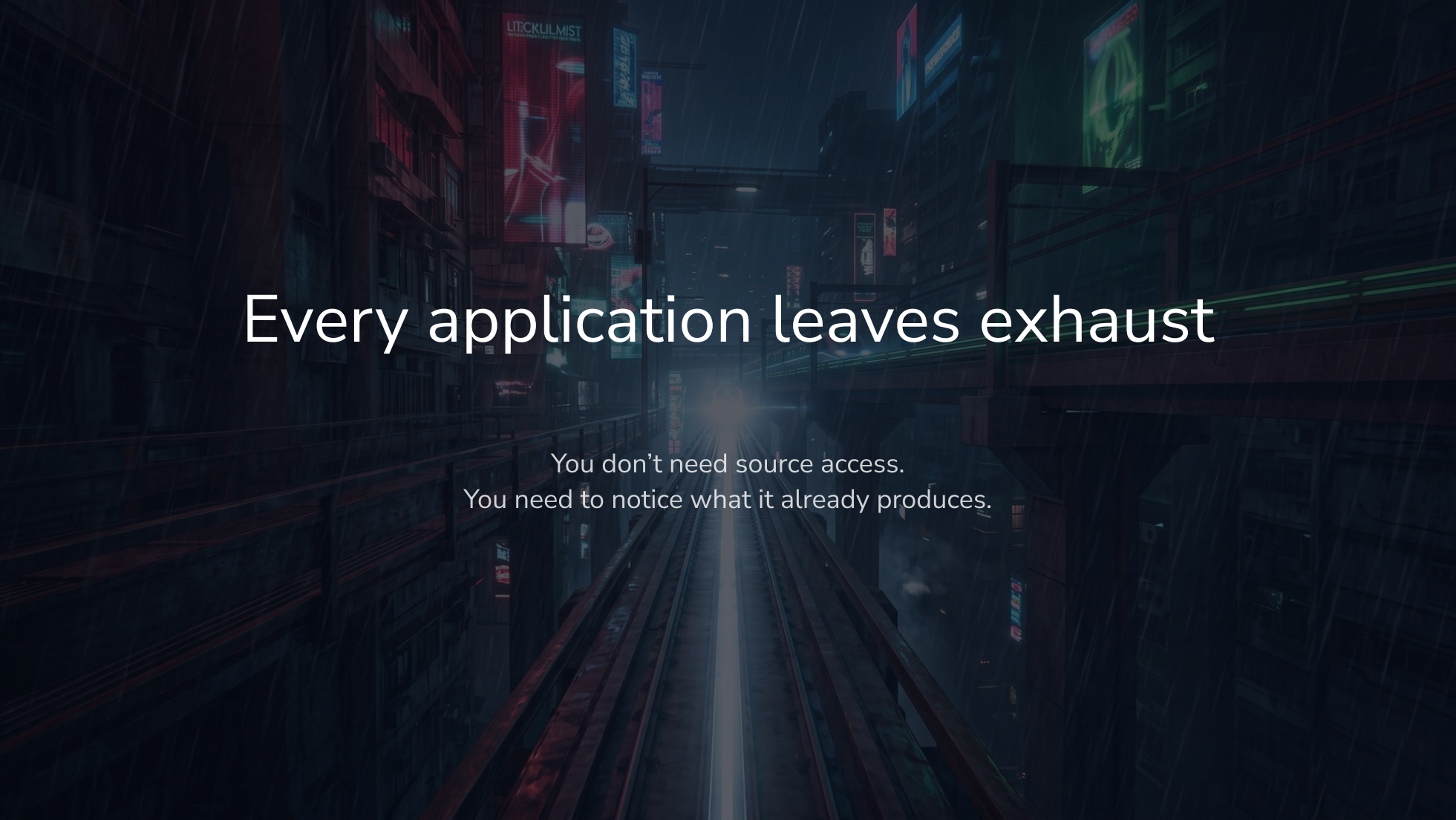
6. Your own application

the one nobody wrote a plugin for



The most important system you run has the worst monitoring

Because the first question everyone asks is
"is there a plugin for it?"



Every application leaves exhaust

You don’t need source access.
You need to notice what it already produces.

The exhaust

It leaves behind	You check	Which proves
A process	`check_process`	it started
A listening port	`check_tcp`	it bound
An HTTP endpoint	`check_http`	it can answer
A log file	`check_logfile`	it’s complaining
Output files	`check_files` *(age)*	it’s still producing
A spool / queue folder	`check_files` *(count)*	it’s keeping up
Registry values	`check_registry_value`	version, state, config drift
A scheduled task	`check_tasksched`	the job actually ran

Not one of these requires the vendor, the source code, or a plugin.

The same four layers, zoomed in

Layer	Check	Proves	Cost	Catches
1	Process running	a PID exists	free	crash
2	Port listening	something bound	free	failed start
2.5	Health endpoint 200	HTTP stack alive	free	hang, deadlock
3	Health endpoint asserted	the app's own verdict	one line	dependency failure, backlog
4	Synthetic transaction	the path works	a script	logic and data errors
4.5	Business metric	it's doing the *right* thing	instrumentation	silent wrongness

Layer 1 is nearly worthless: a hung process is still a process. The cheapest big win is the step from *it answered* to *it says it's healthy*.

Layer 3 is a conversation, not a tool

What you ask a developer for

```
GET /health

{
  "status": "ok",
  "db": "connected",
  "queue": { "depth": 137 },
  "lastRun": "2026-08-06T09:14:00Z"
}
```

A twenty-line endpoint. One afternoon.

What it buys you

```
check_http
  url=http://localhost:8080/health
  "json-path=qlen:queue.depth"
  "crit=qlen > 100"
```

No plugin. No script. No credentials.

The application is the only thing that actually knows whether it's healthy. Everything else is inference.

Demo

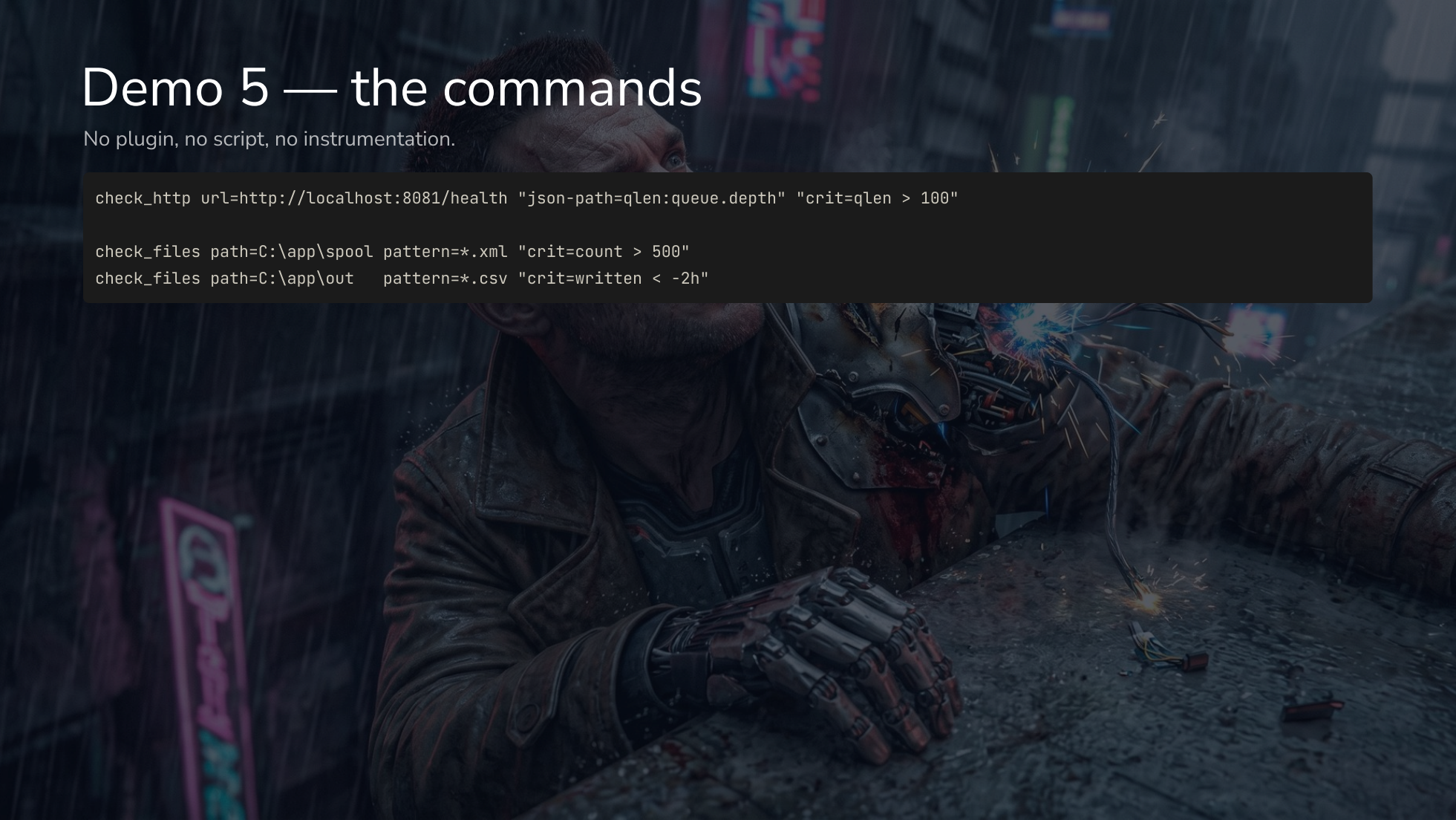
The exhaust, and the endpoint

Demo 5 — the commands

No plugin, no script, no instrumentation.

```
check_http url=http://localhost:8081/health "json-path=qlen:queue.depth" "crit=qlen > 100"

check_files path=C:\app\spool pattern=*.xml "crit=count > 500"
check_files path=C:\app\out   pattern=*.csv "crit=written < -2h"
```



When you do have to script it

Everything above failed? Fine. But know the trap:

```
[/settings/external scripts/wrapped scripts]
check_my_app = check_my_app.ps1
```

Registered under `[scripts]`, a PowerShell script that does `exit 2` arrives as WARNING, not CRITICAL — `powershell.exe` in `-Command` mode exits 0 on success and 1 on any error, regardless of `$LASTEXITCODE`.

`[wrapped scripts]` invokes it with `-File` *so the exit code survives.*

Three more that cost people an evening

- `allow arguments = false` by default. If it's true, anything that reaches the agent port can pass arbitrary arguments to your scripts. Hard-code arguments instead.
- **Payload limits:** NRPE defaults to 1024 bytes, NSCA to 512. Summarise in the message, push detail into performance data.
- **Backslashes double again** through `check_nrpe -a`. What works locally needs escaping from the monitoring server.

7. Getting it to Nagios

and not getting owned on the way

Four routes

		Detection floor
Active — NRPE	Nagios asks, on its schedule	your poll interval
Passive — NSCA / NRDP	agent sends, on its schedule	agent's interval
Real-time	agent sends *when it happens*	seconds
Scrape — Prometheus	for mixed stacks	scrape interval

```
[/settings/system/windows/real-time/process/my_app]
process = my_app.exe
destination = event
critical = state != started
```

Your poll interval is five minutes. Your customer notices in thirty seconds. Almost nobody turns this on.

Don’t get owned

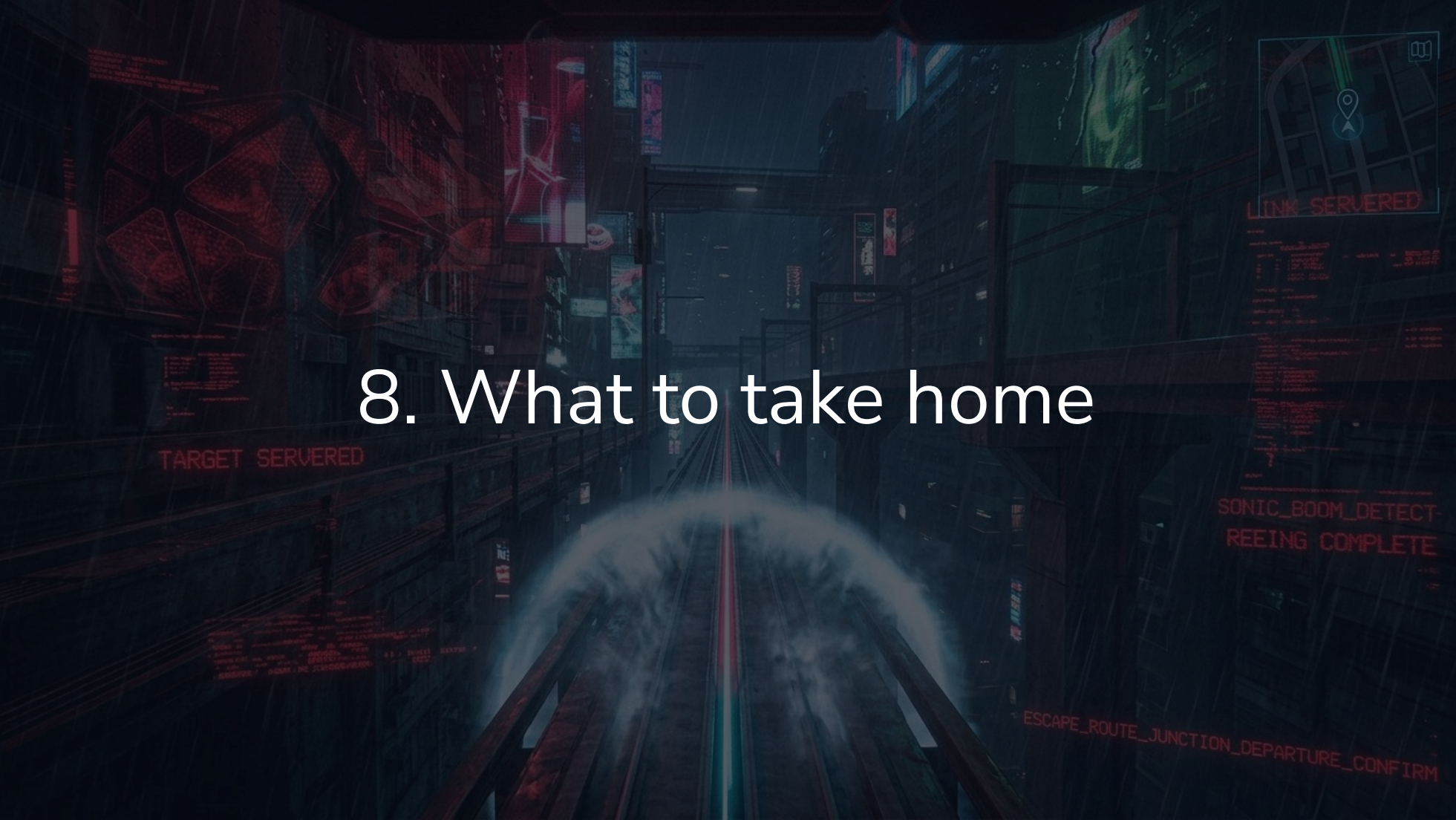
- `allowed hosts` — set it. Not optional. An open agent port is a remote execution service you didn’t mean to run.
- `allow arguments = false` unless you genuinely need it — and prefer hard-coded arguments in the config over pass-through.
- Client certificates on NRPE *(0.12.6+)* — actual identity, not "this IP looks right"
- Encrypt in transit. Monitoring traffic tells an attacker your topology, your hostnames, and which boxes are struggling.

The general principle: a monitoring agent is a remote code execution engine you installed on every server on purpose. Treat it like one.

Monitoring the monitoring

- **Silence is not health.** With passive checks, no news means either "fine" or "the agent is dead" — use freshness/staleness checks to tell them apart.
- **Who watches the monitoring server?** If it's a single box, it's a single point of ignorance.
- **Check that checks still work.** A check pointed at a path that was renamed in 2023 returns OK forever.

> The most dangerous state isn't red. It's **green for the wrong reason.**



8. What to take home

Seven things, none of them a command

1. **Monitoring exists to shorten detection time.** Judge every check by that.
2. **Alert on symptoms, graph the causes.** Windows hands you causes for free and symptoms never.
3. **Walk the four layers.** Existence, reachability, health, function. Layer 4 is where the symptoms are.
4. **Levels tell you where you are; derivatives tell you where you're going.**
5. **Monitor the artifact, not the job.** The output can't lie about itself.
6. **Every application leaves exhaust** — processes, ports, logs, files, queues, tasks.
7. **The best monitoring code lives in your application**, not your monitoring system.

And two that will cost you a day

The agent is not you

LocalSystem. No session, no profile, no mapped drives, no user credentials, different PATH.

Use UNC paths, or run the specific script as a named service account.

PowerShell swallows exit codes

`exit 2` arrives as WARNING unless the script is invoked with `-File`.

Register under `[wrapped scripts]`, not `[scripts]`.

Plus: service state is `'started'`, not `'running'` · PDH counter paths need `\\double\\backslashes` and are localised per Windows language · payload limits are 1024 B on NRPE, 512 B on NSCA

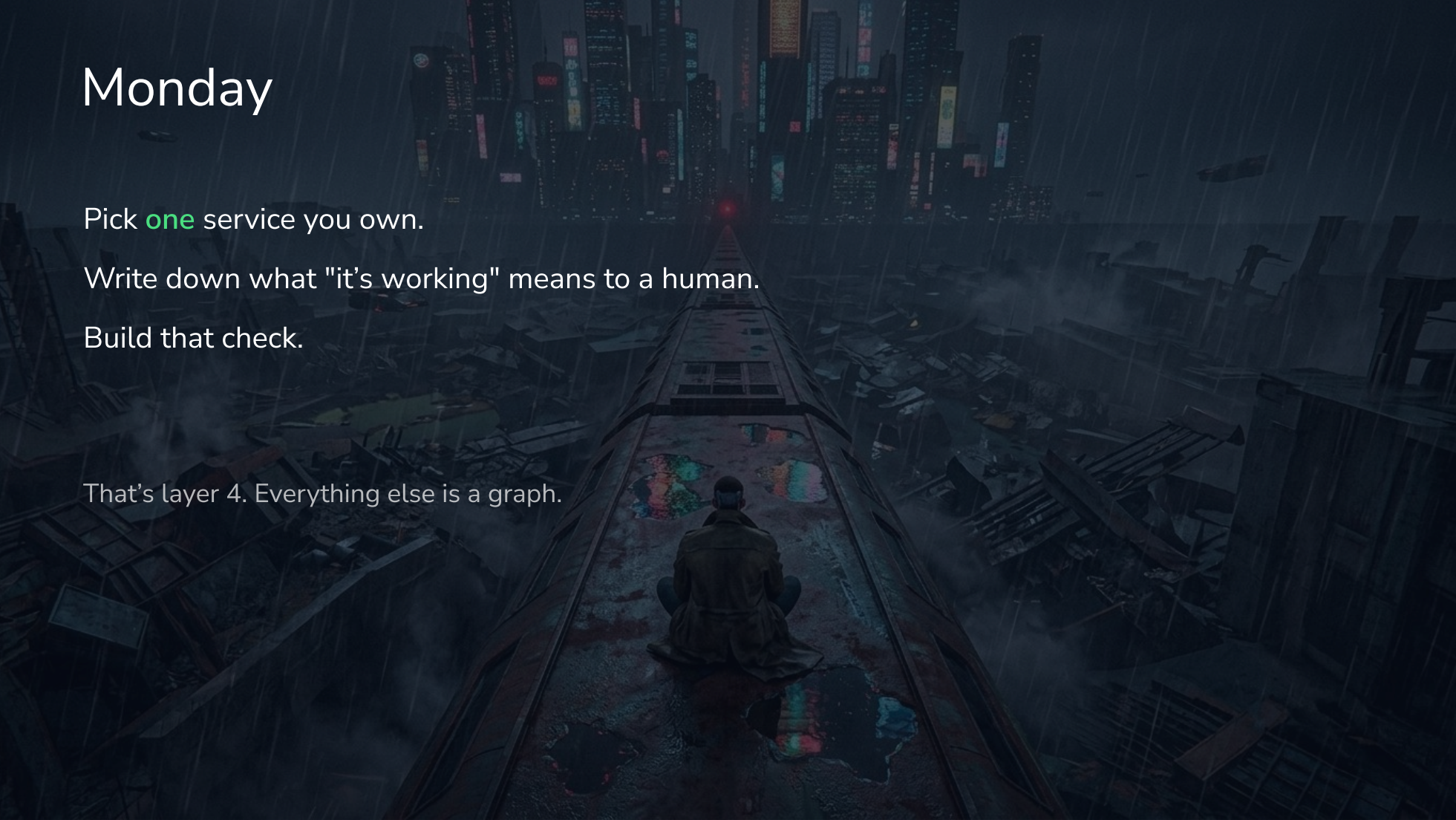
Monday

Pick one service you own.

Write down what "it’s working" means to a human.

Build that check.

That’s layer 4. Everything else is a graph.



Thank you

nsclient.org/docs — the *Monitoring Scenarios* section is step-by-step recipes

github.com/mickem/nscp