Developing for Deterministic Delivery

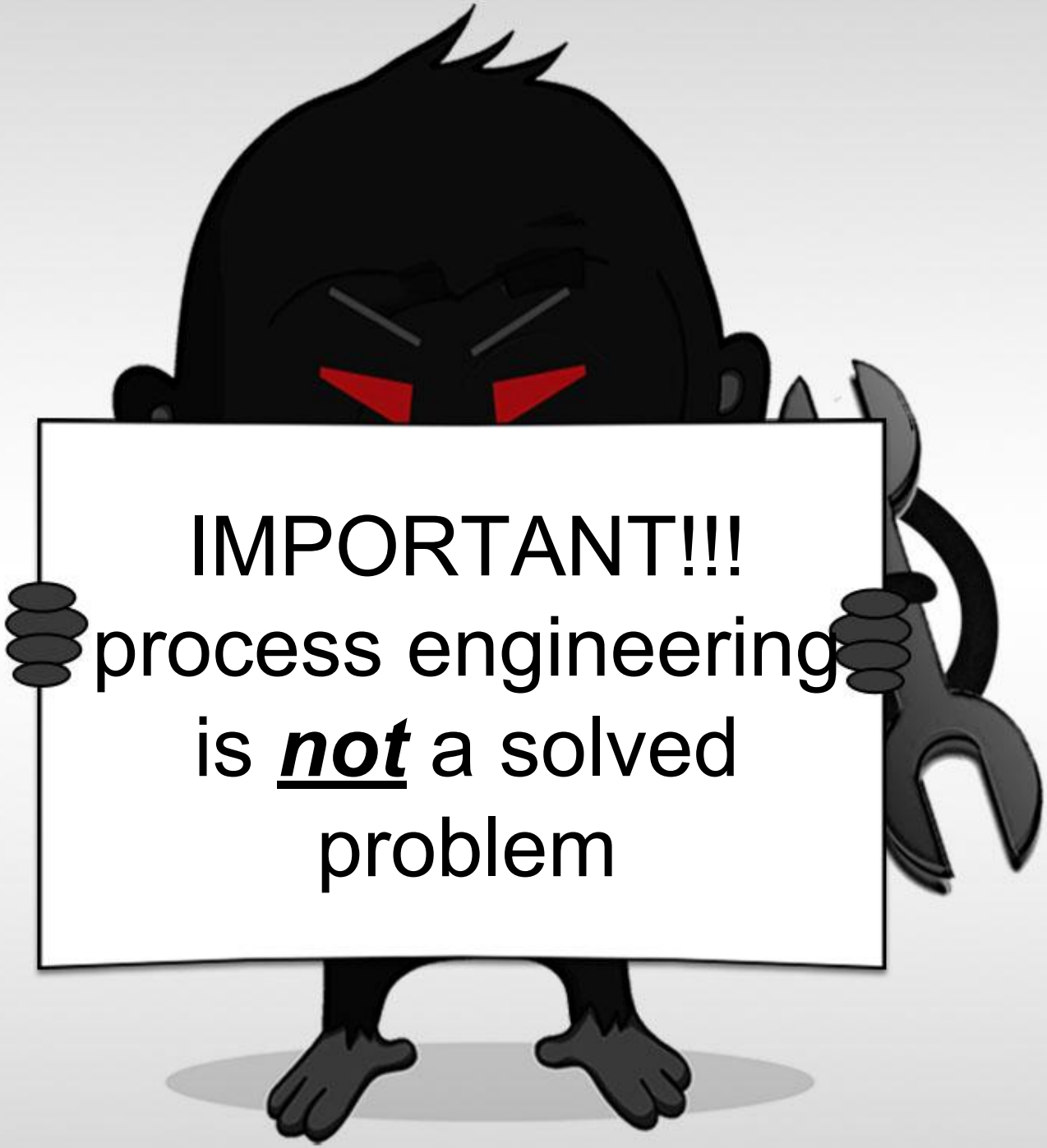
you are lost without a process

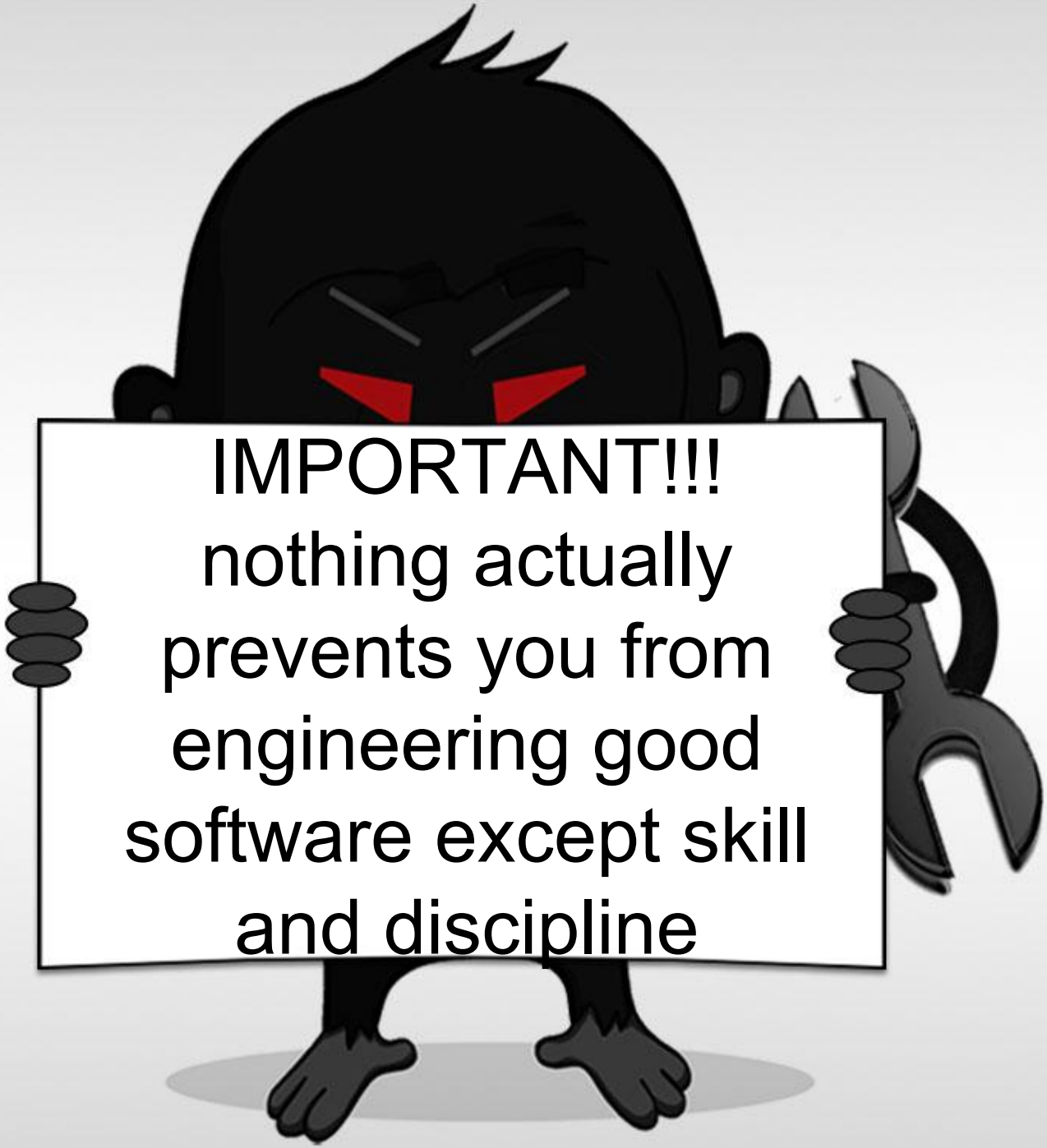
who's this clown

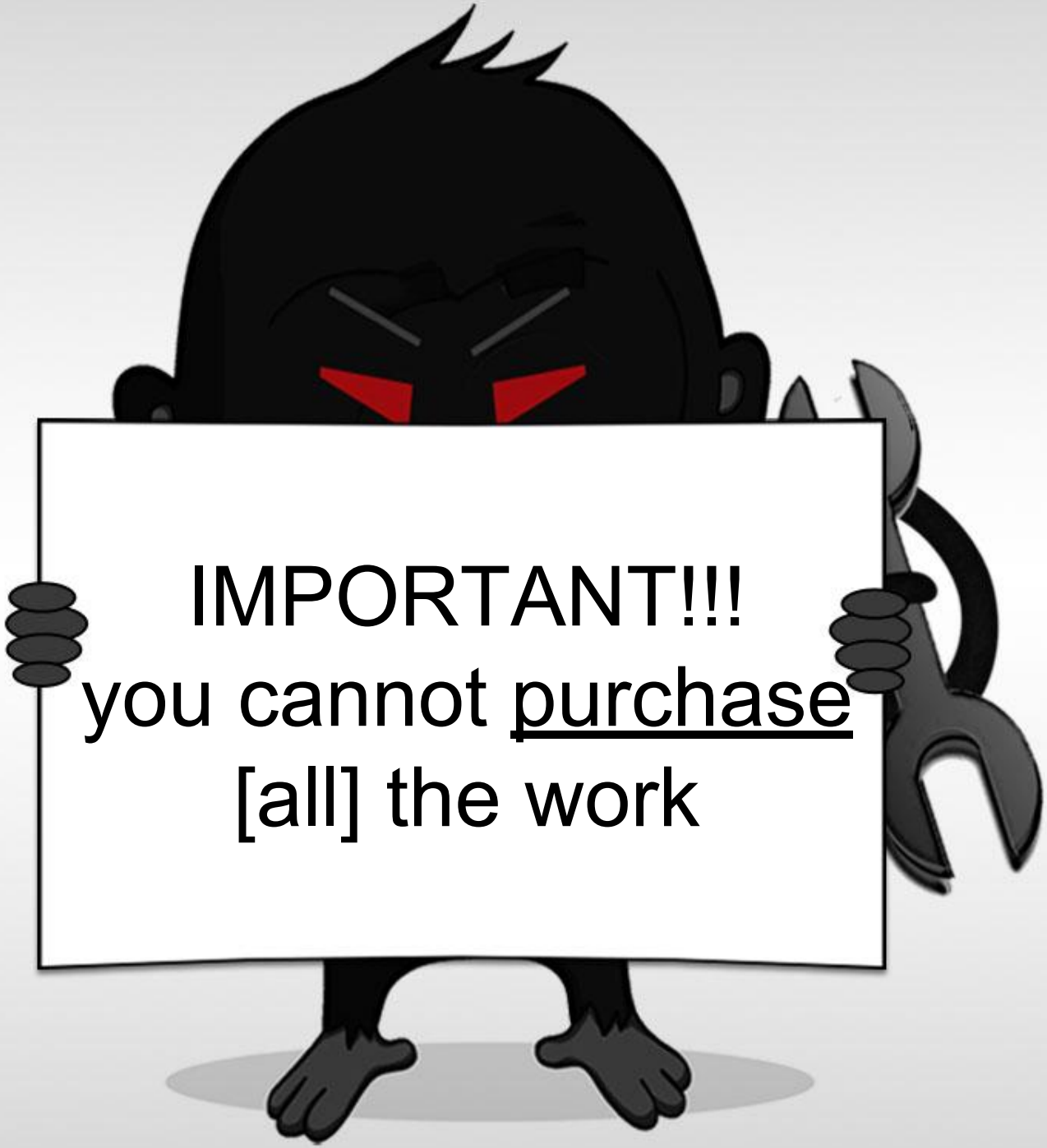
- mykel.alvis (at) gmail.com
- devops coach at Cotiviti Labs
- lawful evil with neutral tendencies
- fixated on deterministic outcomes
- ***probably*** not a shapeshifting, mind-controlling lizardman
- @mykelalvis
- works in a regulated industry

when i look at the matrix i see..

- developers or ops or analysts or testers
- you're all pretty much the same in intent or result
- write code or deploy code or other stuff
- *maybe* you document that stuff, but probably not
- *maybe* you realize how hard your stuff is to deal with, but probably not


IMPORTANT!!!
process engineering
is ***not*** a solved
problem


IMPORTANT!!!
nothing actually
prevents you from
engineering good
software except skill
and discipline


IMPORTANT!!!
you cannot purchase
[all] the work


digging up
the lede

[tl];dr ftw!

- provide very limited assurances
- make an inoffensive architecture
- apply process
- apply even more testing
- produce immutable results
- deal with dependency management
- execute continuous improvement
- use your immutable results

[tl];dr ftw! 2 electric boogaloo

- a formal *release* process
- using *controlled* source code
- with *managed* dependencies
- and *versioned* builds
- that are *tested* AF
- and *continuously* integrated
- producing *immutable* artifacts
- that you keep *forever**

the way™

- write your code
- unit and integration test your code
- identify your code (version it)
- note that identification (make a tag!)
- checkout that tag and make an artifact
- save that artifact
- only use that artifact
- repeat

mr webster is rolling over

- artifact
  - some thingy or another, produced somehow
  - a metaphorical bucket of bytes
- platform
  - somewhere that you want some artifact to be available to its consumer
- environment
  - some *precisely* configured platform
- availability
  - the state of being accessible to a consumer

mr webster is still rolling over

- version
  - identifying *signature* of an artifact
- build
  - produce a *potential* artifact
- release
  - build of an *immutable* version of an artifact
- deployment
  - induce *availability* of an artifact ***in an environment*** *(see what I did there?)*

the pedantic view from 30,000 feet

- everything in your life is or has a process
- all process has a flow
- all flows change over time
- jobs are executions of these flows that we get paid to execute
- we use process to ***increase*** more of the *-ilities*

pedantics...qualified

- just because there's a fixed process doesn't mean that it always executes the same way every time
- because chaos is omnipresent
- and chaos is not your friend
- but neither is it necessarily your enemy
- it shows your the edges of your system

keep rollin', rollin', rollin'

- determinism (not *precisely* the philosophical definition)
  1. ~~the belief that all events are caused by things that happened before them and that people have no real ability to make choices or control what happens~~
  2. a theory or doctrine that acts of the will, occurrences in nature, or social or psychological phenomena are causally determined by preceding events or natural laws

From http://www.merriam-webster.com/dictionary/determinism


today is the
result of all the
yesterdays

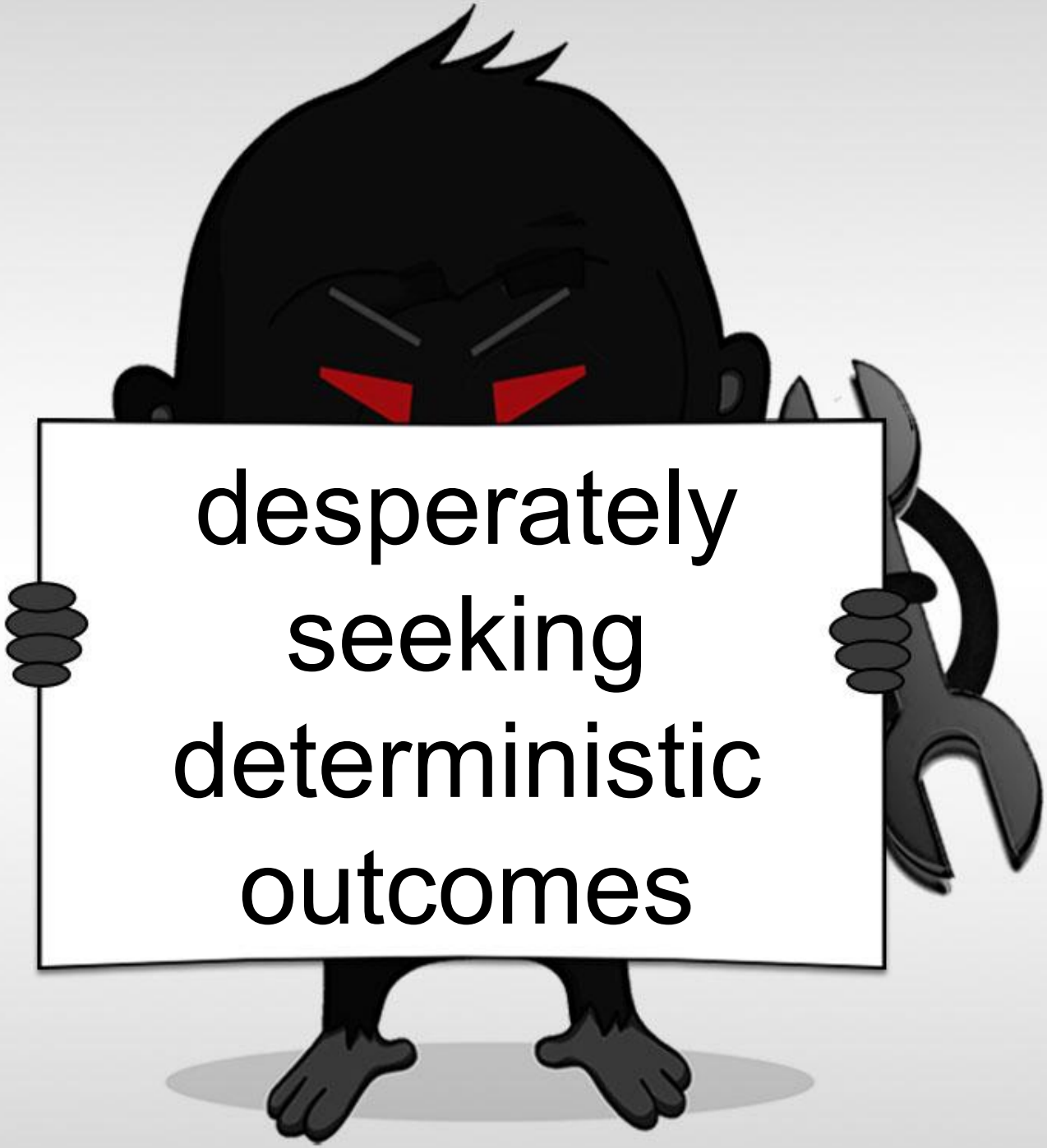
so?

- your stuff is prolly complicated
- your stuff is prolly error-prone
- it prolly has more steps that it should
- it is prolly difficult to replicate
- it is nigh-certainly human-centric
- ***delivering your stuff to the customer is literally the only valid job in any enterprise and frequently is not done well***


ur process
might be
teh dum

the goal

- i want to play video games all day
- i only want to be disturbed if something is ***actually wrong***
- automated deployment
- automatic maintenance of the deploy
- documentation back to source
- reliability
- repeatability
- auditability


desperately
seeking
deterministic
outcomes

whazzat really mean?

- "***deterministic***" implies "***predictable***"
  - well, that's how i'm using it here
- "***predictable***" actually means "***somewhat reasonably predictable***"
  - within constraints of "reasonable"
- "***reasonably predictable***" is pretty h*ckin' good


actual
determinism
is impossible

so how do we do it?

- we're trying to solve
  - an impossible problem
  - and produce an impossible result
  - with deficient tools
  - held to an oft-unreasonable standard by the enterprise
- how can we approach this problem?
- how did mykel do it?


source code
control

goin' straight to the source

- source code control
- use it or seriously be considered a fool
- learn to do merges properly
- establish a reasonable scm workflow
- which tool you use is ***way*** less important than the workflow you use


immutability

to serve man

- there are many mechanisms to handle this problem
- some of them are better than others, for various variances on the term “better”
- ymmv

and relative dimensions in space

- tags are a pretty easy way
  - `git tag -a homeslice-1.0.0 -m "Taggin the homeslice"`
  - volatile in git/hg, but not subversion
- tags let you talk about *specific points in time* via source control
- people who don't tag their code piss me off

buckets o’ code

- containers [+ [micro-]services]
- easily deployed
- ok to configure and connect
- h*cking pain to maintain
- continue to improve as a technology

tilling the brown fields

- managing a container deployment can be pretty difficult
- getting way better
- kubernetes is The Way™

old_kit_bag.pack(troubles)

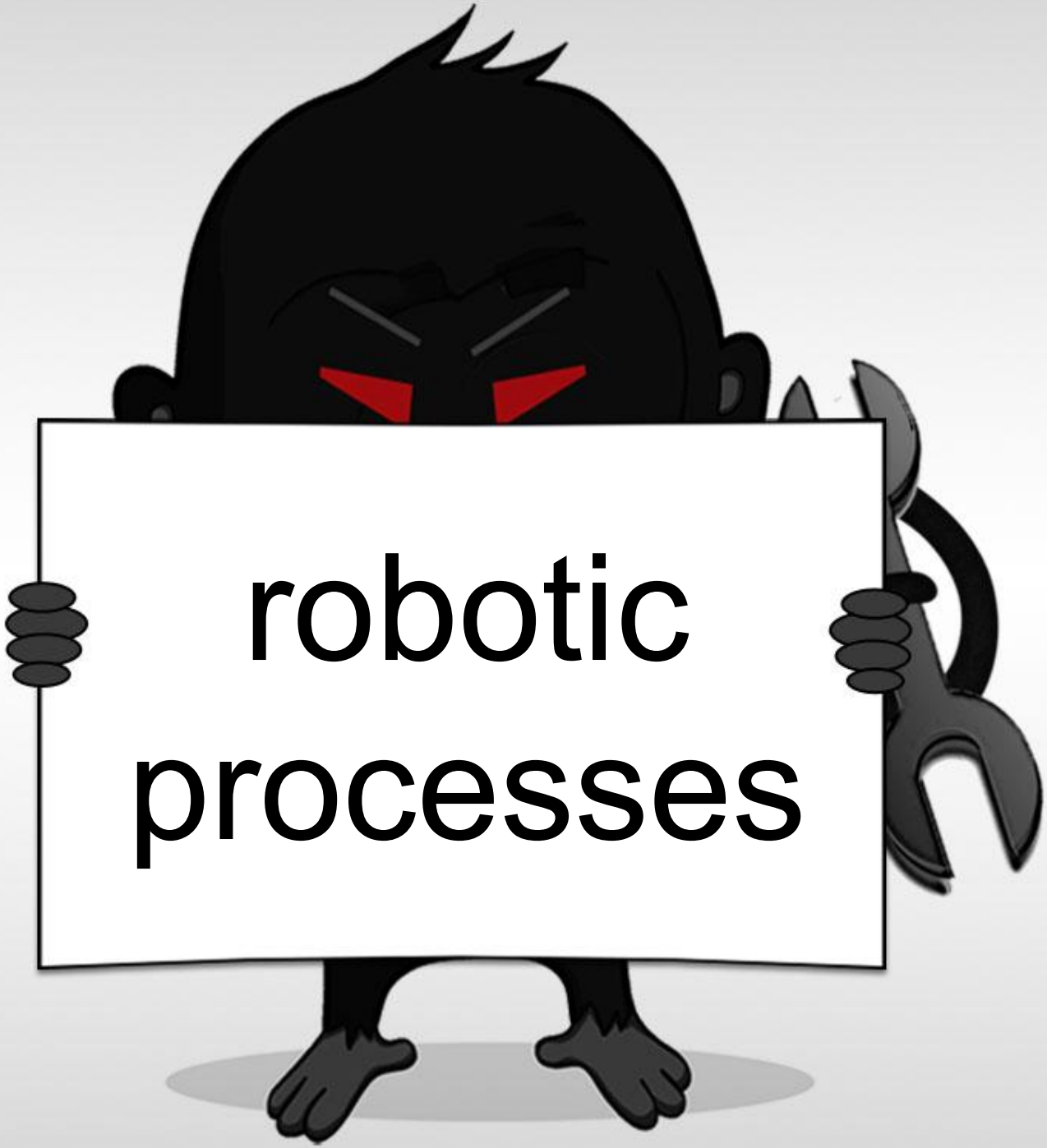
- external management systems
- SaaS/PaaS/IaaS
- Cloud Formations
- Atlas et al
- marriage (i.e typically a long-term thing)

perversions of science

- some twisted, proprietary mechanism
  - (it's sort of what I'm doing)
- definitely works for you
- **definitely** challenging to maintain
- again, marriage
- this time your spouse is a monster

well, then how do we do this?

- no one wants the responsibility
- no one *really* wants the authority
- the business always wants accountability
- so take all of it away from humans
- give your delivery over to a fixed process


robotic
processes

#ffs elsa, let it go

- hand the responsibility over to a robot
- hubot and the like are fine candidates
- robots are developed just like other code
- bootstrapping is a good approach

don’t you ever change

- immutable != stagnant
  - allows ignoring change requests for existing things
  - intrinsically repeatable (*by my definition!*)
- any given system must be stable
  - make it stable
  - if it needs to change, replace it
  - but what to replace it with?


versioning


specifically
http://semver.org

and relative dimensions in space pt2

- replace old versions with a new version
  - duh
- semantic versions
  - a view of namespaces into functionality
  - specify api compatibility
- provide a good fall back

the next big thing

- versioned environments tied to deploys
- *deployment* of a versioned artifact is itself versioned
- versioned deployments are trackable
  - (remember that versions imply releases)
- but even versioned deployments can be arbitrarily complex
- somewhat unsolved: how do we address the complexity of the deployment itself?


dependency
management

think of the kittens

- dependency management is 50-85% of your job
- and you probably aren’t doing it very well
- so h*cks sake, use a dependency caching-proxy
  - nexus
  - artifactory
  - gem/geminabox
  - ANYTHING!


but why,
mykel?

duplo blocks: architecture 098

- separation of concerns is a concern
- looser coupling is harder to write and easier to iterate on
- refactoring is always less work than re-writing
  - but might not be as fun
  - ask your boss if you get paid for the fun

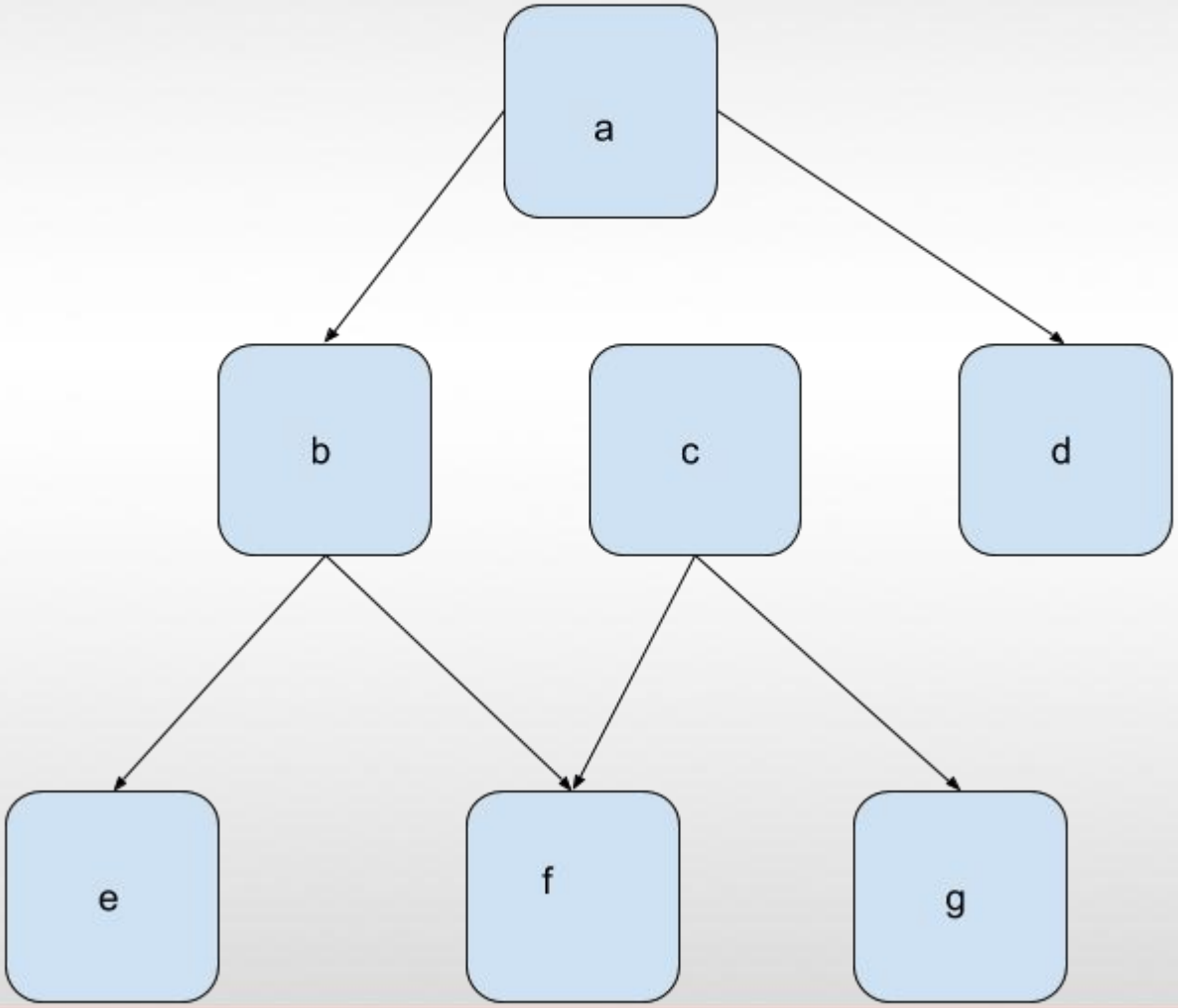
legos: architecture 101

- arbitrary complexity must be encapsulated
- encapsulated complexity must be versioned
- versioned complexity must be orchestrated
- and tracked
- and audited
- and blah blah blah
- in short, and to use a dirty word in the devops space, ***enforced***

there's stormtroopers comin', pa!

- enforcement doesn't mean being a h*kin a-hole
  - unless it does, then be one
- you must decide *which set of things* you want to be deterministic
- the *costs* of determinism can be quite high
- you need to get the process internalized into the executors

how your life works

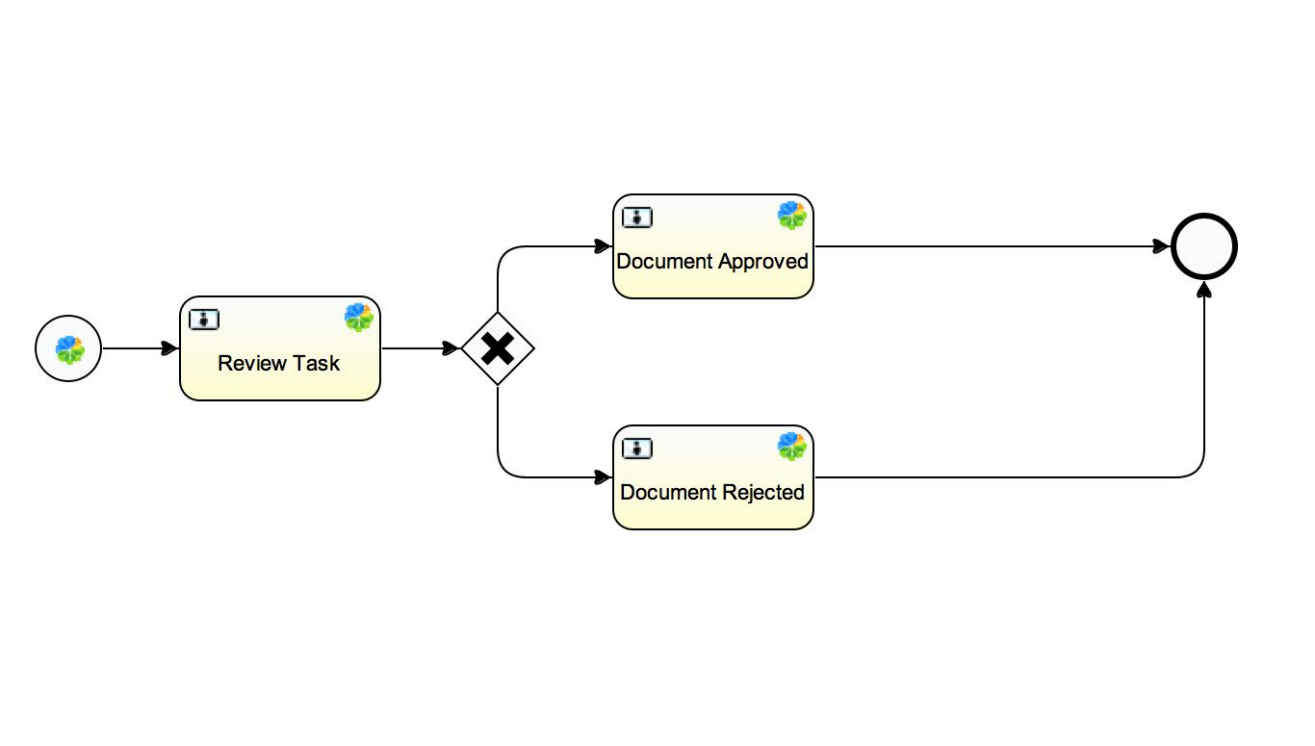


A. wake
B. bathe
C. clothes
D. coffee
E. food
F. car
G. work

i got flow like marshall mathers

- workflows allow orchestration
- engines running it are typically solid
  - they can be kinda clunky to deal with
  - and they're like democracy
- design to be as linear as possible
- jenkins works just fine for beginner workflows

what does it look like



what does it really look like

```
41          <activiti:field name="script">
42             <activiti:string>
43                if (typeof bpm_workflowDueDate != 'undefined') task.setVariableLocal('bpm_dueDate', bpm_workflowDueDate);
44                if (typeof bpm_workflowPriority != 'undefined') task.priority = bpm_workflowPriority;
45             </activiti:string>
46          </activiti:field>
47       </activiti:taskListener>
48       <activiti:taskListener event="complete" class="org.alfresco.repo.workflow.activiti.tasklistener.ScriptTaskListener">
49          <activiti:field name="script">
50             <activiti:string>
51                if(task.getVariableLocal('wf_reviewOutcome') == 'Approve') {
52                     var newApprovedCount = wf_approveCount + 1;
53                     var newApprovedPercentage = (newApprovedCount / wf_reviewerCount) * 100;
54
55                     execution.setVariable('wf_approveCount', newApprovedCount);
56                     execution.setVariable('wf_actualPercent', newApprovedPercentage);
57                }
58             </activiti:string>
59          </activiti:field>
60       </activiti:taskListener>
61    </extensionElements>
62
63    <humanPerformer>
64         <resourceAssignmentExpression>
65            <formalExpression>${reviewAssignee.properties.userName}</formalExpression>
66         </resourceAssignmentExpression>
67    </humanPerformer>
68
69    <!-- For each assignee, task is created -->
```

nothing is ever enough

- but that was more than just a bit clunky
- versioning wasn't impossible,
  just [too] hard
- automated deployment was [quite] hard
- the results weren't deterministic enough
  [for me]

straight on til kandahar

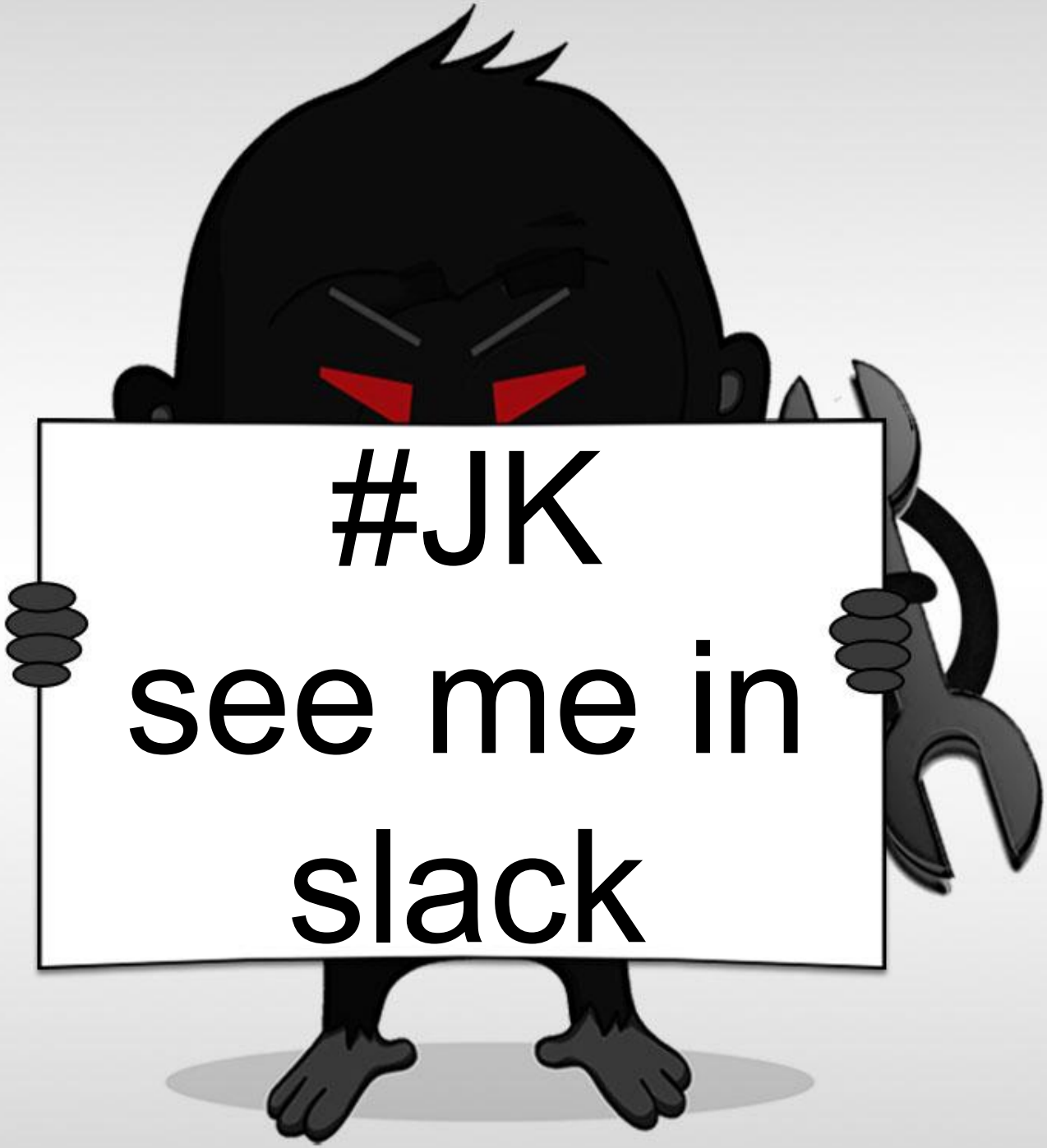
- build a versioned model of a system
- “realize” the model (this is the exceptionally tricky part)
- profit!

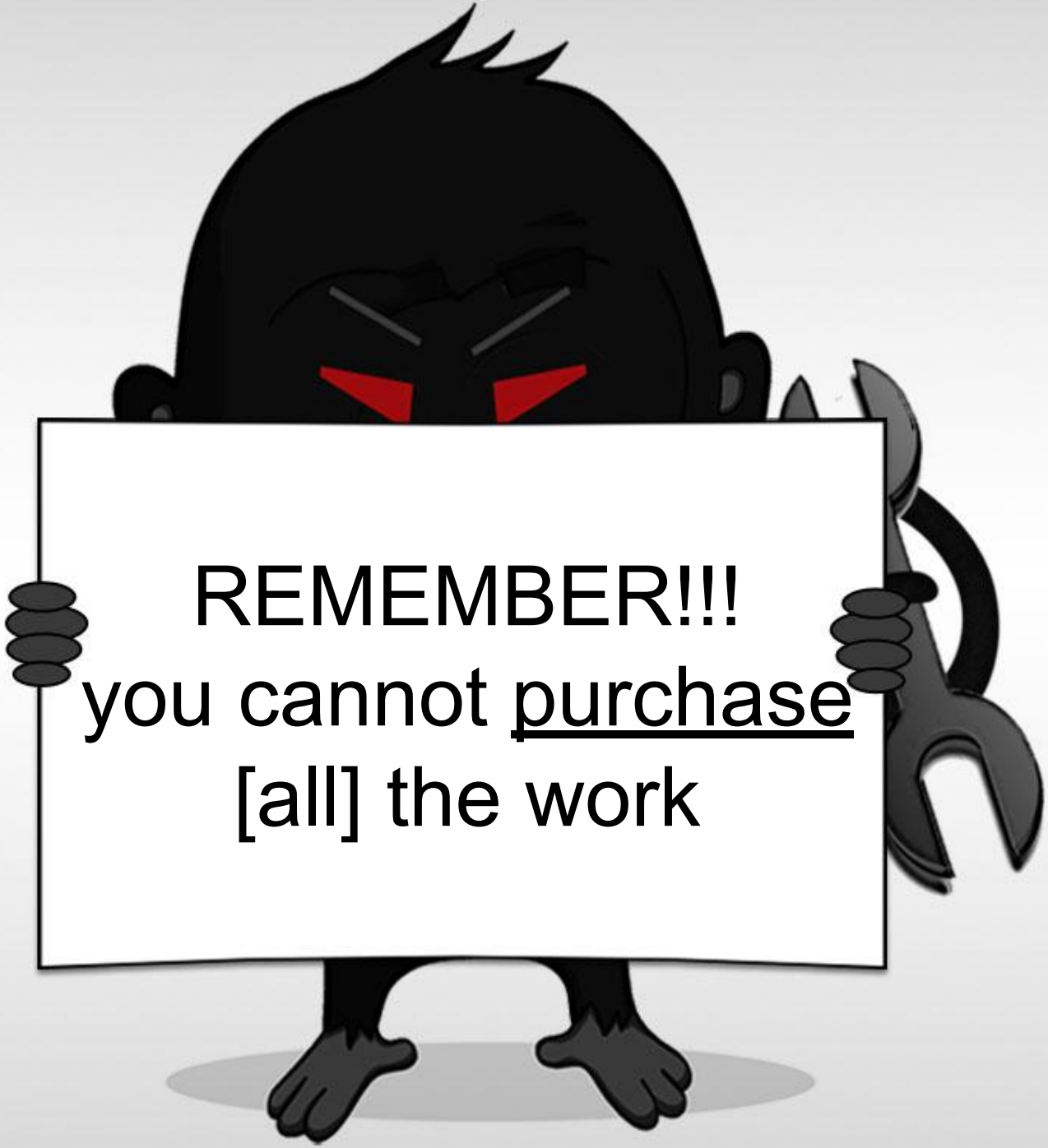

demo?


any more
questions?


no?


good


#JK
see me in
slack


REMEMBER!!!
you cannot purchase
[all] the work


ALSO:
DOCUMENT
YOUR CRAP!


namaste
puppers!
So long and thanx for all the lulz
@mykelalvis