PolicyKit

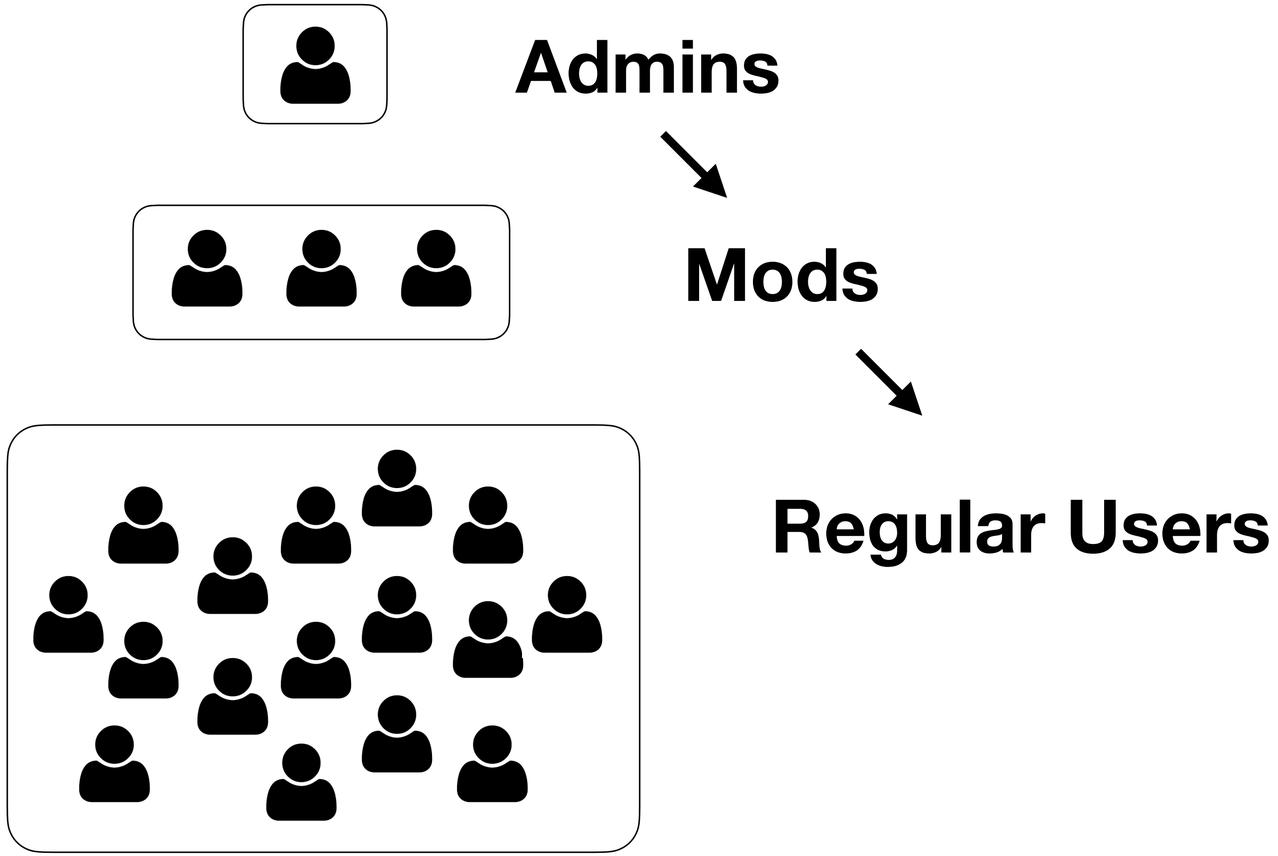
Building Governance in Online Communities

Amy X. Zhang[1,2] axz@cs.uw.edu
Grant Hugh[2] ghugh@stanford.edu
Michael Bernstein[2] msb@cs.stanford.edu

[1]University of Washington
[2]Stanford University

Consider all the tools you use to participate in online communities today.

When it comes to the governance model that these tools provide to communities, they all share a strikingly similar pattern:



These tools describe governance using a ***permissions model***. So when a user wants to do something, the tool just checks what ***permissions*** they have before they can do it.

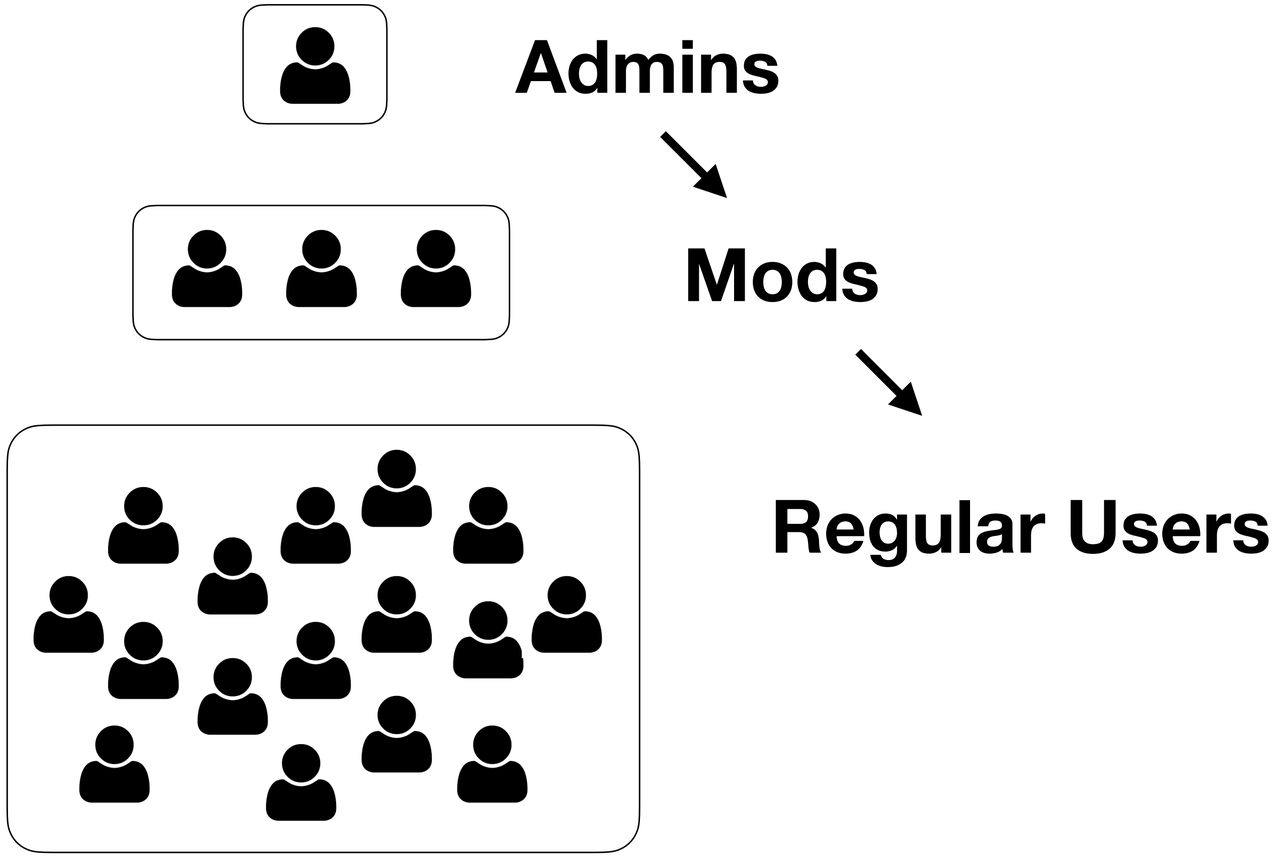
User A would like to do Action X

check their permissions

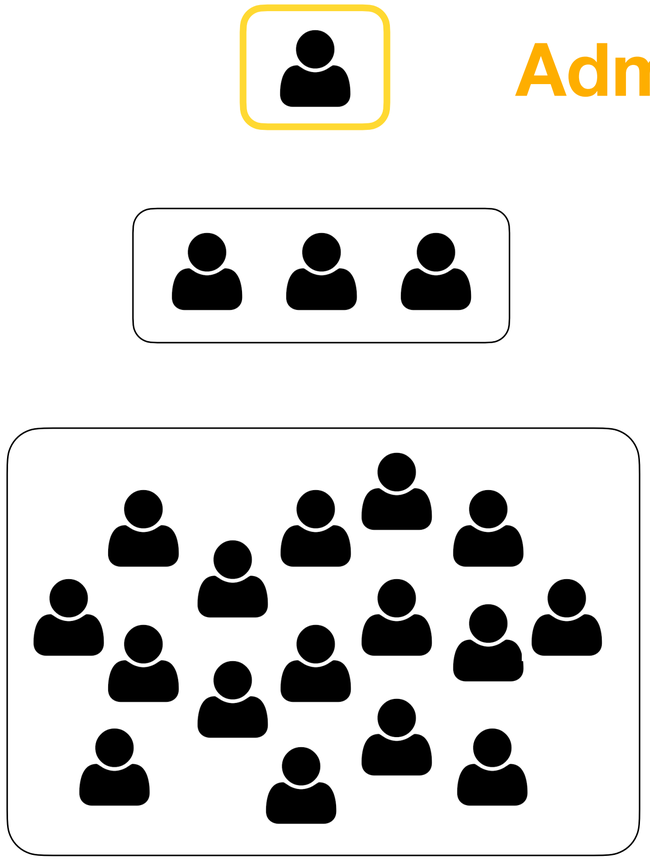
	Action X	Action Y	Action Z
User A	✓		
User B			

Action X is approved

And in order to **modify governance**, only certain ***roles*** (like an admin) can change a user's permissions.



And in order to **modify governance**, only certain roles (like an admin) can change a user's permissions.



Admins

Mods

Regular Users

Meanwhile regular users have no say in the governance of their community.

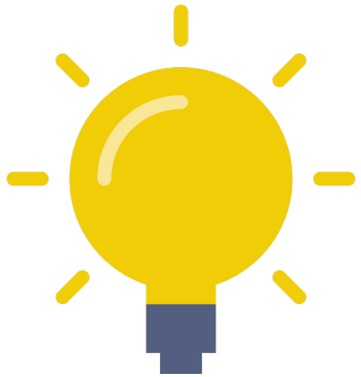
This **permissions model** is also prevalent in most collaborative software that you use everyday, even down to the access control for files in UNIX-like operating systems.

A permissions model of governance can only describe a limited range of governance models—mostly top-down, autocratic, and punitive ones.

If a community wants to have a different style of governance, like a democratic system for making decisions, they have to either:

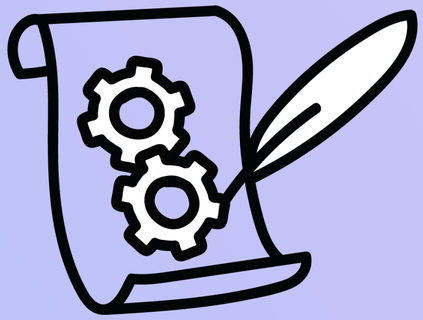
- carry it out manually in an ad hoc way
- use or build one-off software tools

cumbersome
error-prone
hard to maintain



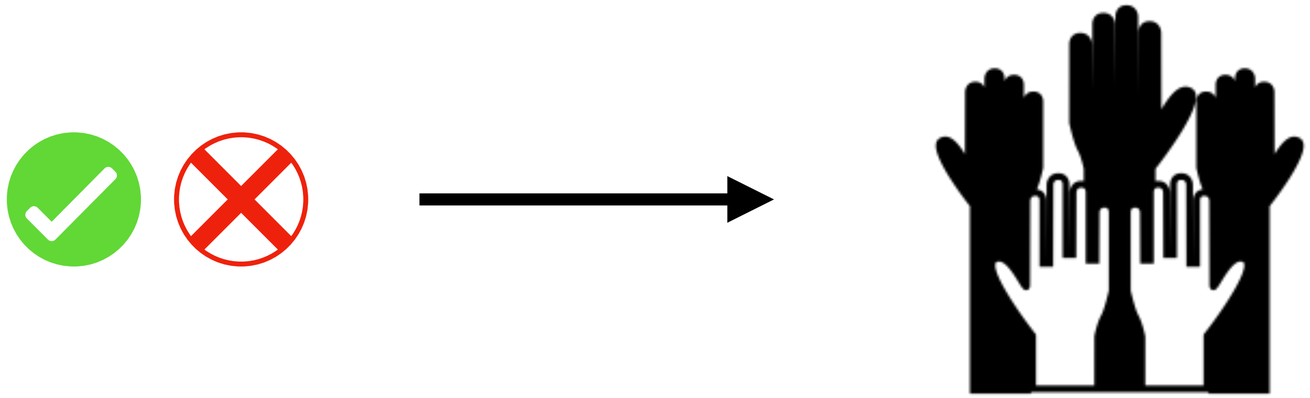
But what if there was a way to express other forms of governance in the software, such as more democratic models?

What if communities had a way to build for themselves the governance that suits their needs and values?



PolicyKit

a software infrastructure that empowers online community members to **concisely author** a wide range of governance procedures and **automatically carry out** those procedures on their home platforms



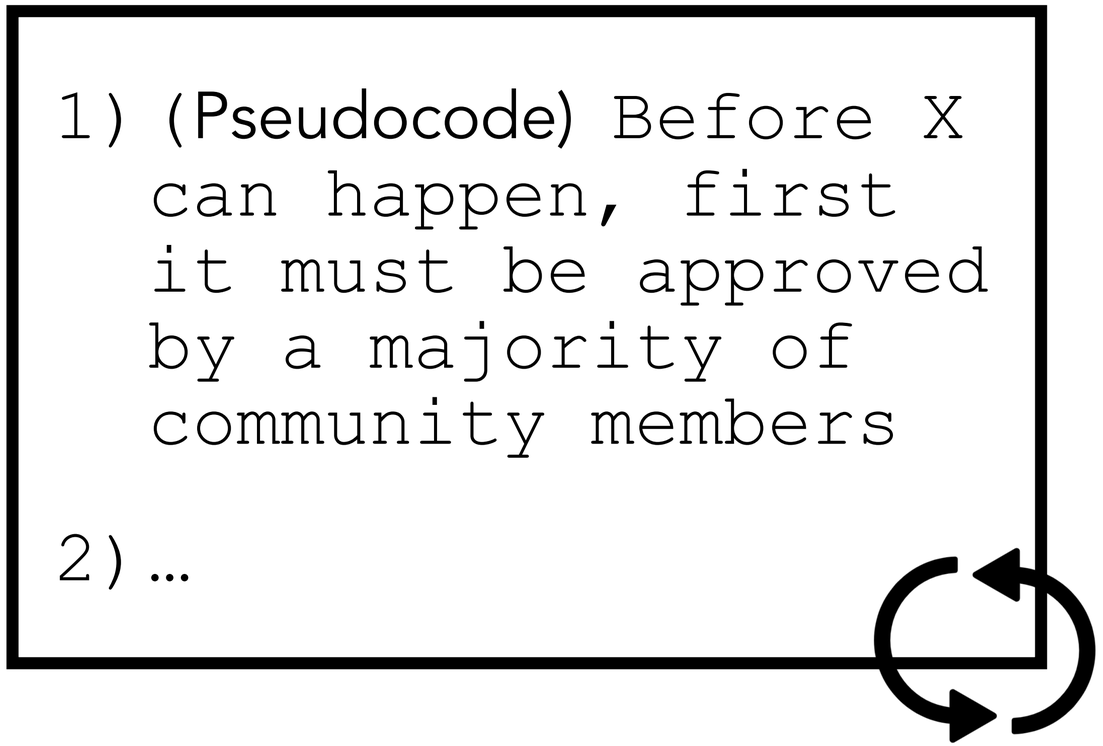
PolicyKit's main insight is to shift governance from articulating ***permissions*** to articulating ***procedures***, where procedures can express a wide range of governance models concisely, including participatory and democratic models.

So instead of looking up a user's permissions, we instead look up and run **policies** in PolicyKit, or short procedural scripts, that govern an action.

User A would like to do Action X

Action X is approved

check the policies governing X



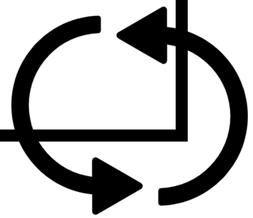
Users can also write policies to govern actions such as adding or updating a policy, allowing users to participate in **modifying governance**.

User A would like to change the policy governing X

check the policies governing changing existing policies

1) (Pseudocode) To change any policy, a two-thirds majority vote must first approve the change.

2) …



The policy is changed

We call policies that govern everyday actions on a community platform **Platform Policies** and

policies that govern actions to modify the governance itself **Constitution Policies**,

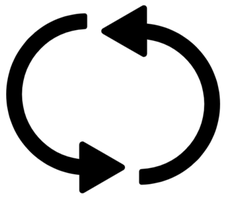
taking inspiration from political science theory developed by Elinor Ostrom.

Examples

Actions: one-off events that users can propose

Sara would like to post the message “Hi” to the #announcements channel on Slack.

Platform



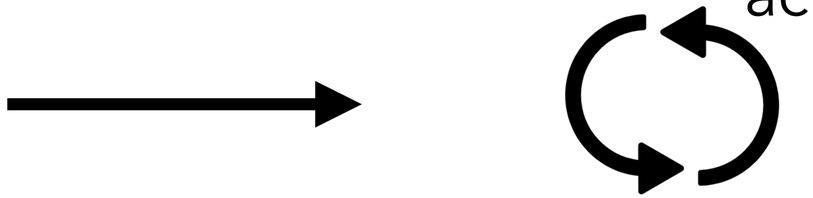
Policies: continually running scripts that govern actions that come up

Any posts to the #announcements channel must first be approved by a moderator.

Examples

Actions: one-off events that users can propose

Policies: continually running code declarations that govern actions that come up



Sara would like to post the message “Hi” to the #announcements channel on Slack.

Platform

Any posts to the #announcements channel must first be approved by a moderator.

Jane would like to introduce a new policy for renaming channels.

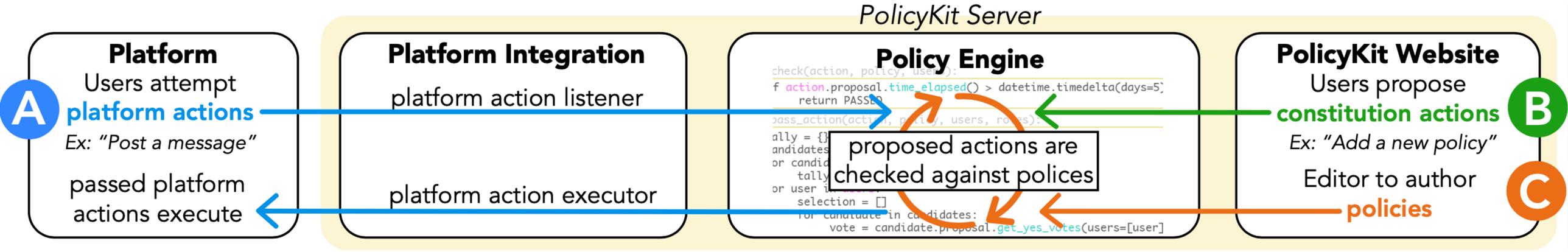
Constitution

To add any new policy, it must first be voted in by majority vote.

PolicyKit Components

- platform integrations
- policy engine
- PolicyKit website

How they all comes together:



Following is an example of a policy that creates a random jury of members to vote on renaming Slack channels.

Writing a policy involves implementing 6 functions.

The policy filters to only consider actions involving renaming Slack channels.

Filter

```
def filter(action, policy):
    if action.action_type == 'SlackRenameChannel':
        return True
```

Then we initialize the policy by creating a random jury of 3 community members to vote on this rename.

Initialize

```
def initialize(action, policy):
    usernames = [u.username for u in users]
    jury = random.sample(usernames, k=3)
    action.data.add('jury', jury)
```

We notify those 3 jury members about what they're voting on and other instructions.

Notify

```
def notify(action, policy):
    jury = action.data.get('jury')
    jury_users = users.filter(username__in=jury)
    action.community.notify_users(action, policy, users=jury_users,
        text='Please deliberate amongst yourselves before voting')
```

Here we check whether an action can pass. If 2 or more jury members have voted yes - then it passes. If 2 days have gone by without it passing, then it fails.

Check

```
def check(action, policy):
    jury = action.data.get('jury')
    jury_users = users.filter(username__in=jury)
    yes_votes = action.proposal.get_votes(users=jury_users, value=True)
    if len(yes_votes) >= 2:
        return PASSED
    elif action.proposal.time_elapsed() > datetime.timedelta(days=2):
        return FAILED
```

The channel gets renamed on Slack.

Pass

```
def pass_action(action, policy):
    action.execute()
```

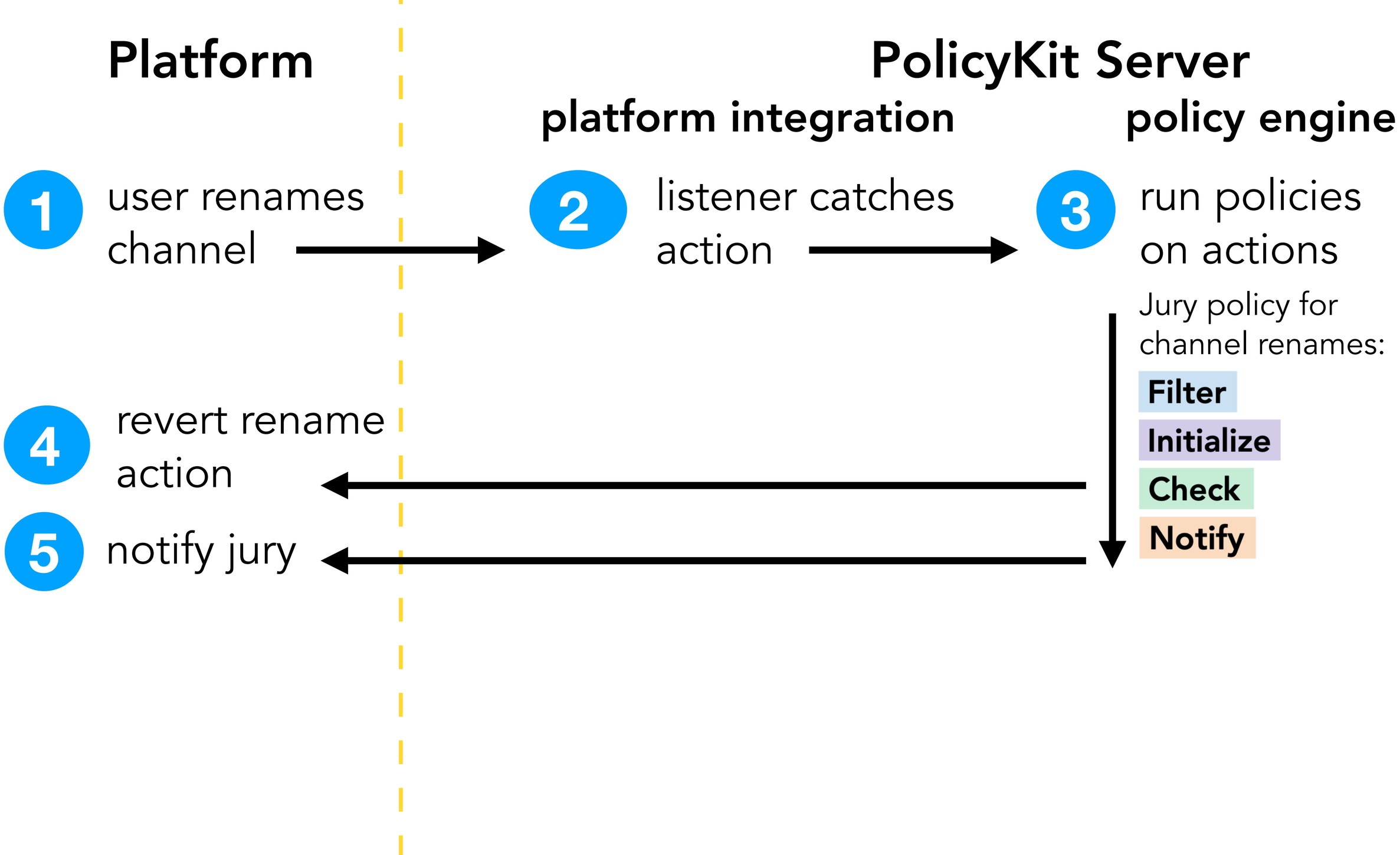
It failed so nothing happens.

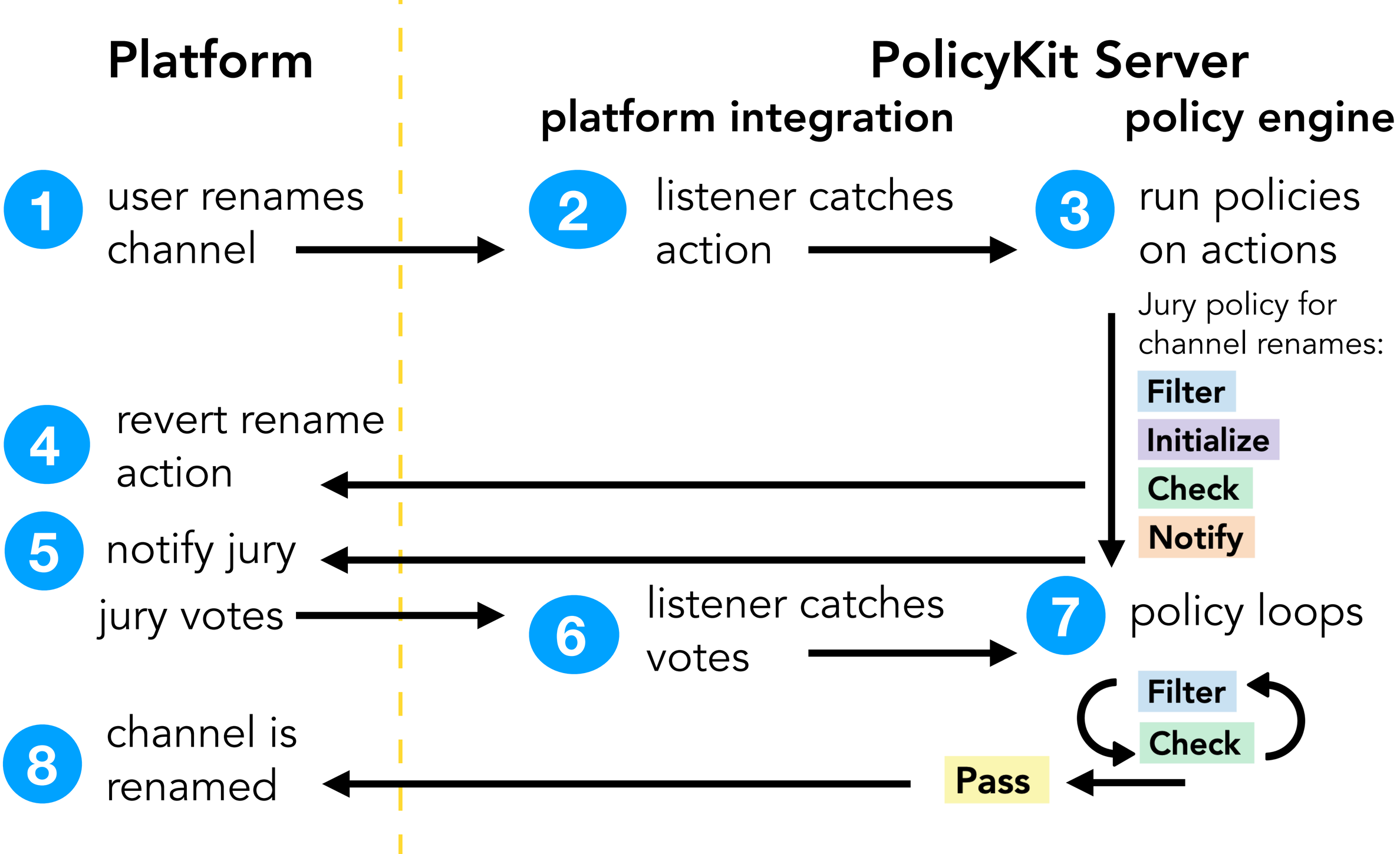
Fail

```
def fail_action(action, policy):
    return
```

PolicyKit:

a software infrastructure that empowers online community members to **concisely author** a wide range of governance procedures and **automatically carry out** those procedures on their home platforms


Platform
PolicyKit Server
platform integration
policy engine
1
user renames channel
2
listener catches action
3
run policies on actions
Jury policy for channel renames:
Filter
Initialize
Check
Notify
4
revert rename action
5
notify jury


Platform
PolicyKit Server
platform integration
policy engine
1
user renames channel
2
listener catches action
3
run policies on actions
Jury policy for channel renames:
Filter
Initialize
Check
Notify
4
revert rename action
5
notify jury
jury votes
6
listener catches votes
7
policy loops
Filter
Check
Pass
8
channel is renamed

PolicyKit API

JSON blob data with each action and policy to be able to have memory

Initialize

```
def initialize(action, policy):
    usernames = [u.username for u in users]
    jury = random.sample(usernames, k=3)
    action.data.add('jury', jury)
```

Notify

```
def notify(action, policy):
    jury = action.data.get('jury')
    jury_users = users.filter(username__in=jury)
    action.community.notify_users(action, policy, users=jury_users,
        text='Please deliberate amongst yourselves before voting')
```

Extensions to the Basic Data Model

roles and permissions that override policies

text documents for natural language policies like community guidelines

action bundles for elections (choice between multiple actions)

policy bundles for multi-stage policies (like a two-round caucus)

UDD0ZNYMS@tes...

APPLICATIONS

POLICYENGINE

- Boolean votes
- Communitypolicybundles
- Number votes
- Policykit add community policys
- Policykit add permissions
- Policykit add process policys
- Policykit add roles
- Policykit add user roles
- Policykit change community docs
- Policykit change community policys
- Policykit change process policys
- Policykit delete roles
- Policykit remove community policys
- Policykit remove permissions
- Policykit remove process policys
- Policykit remove user roles

PASSED PROCESS POLICIES

Starter Policy: all policies pass

PASSED COMMUNITY POLICIES

majority rule to pass any community policies

Filter Code:

```
return True
```

Init Code:

```
pass
```

Notify Code:

```
post_policy(policy, action)
```

Conditional Code:

```
aye_user_votes = UserVote.objects.filter(proposal=act
num_in_community = CommunityUser.objects.filter(commu
import math
if aye_user_votes >= math.ceil(num_in_community/2.0):
  return Proposal.PASSED
```

Action Code:

```
execute_action(action)
```

Failure Code:

```
pass
```

ROLES AND PERMISSIONS

Role: Base User

Permissions:

- Can add boolean vote
- Can change boolean vote
- Can delete boolean vote
- Can view boolean vote
- Can add communityactionbundle
- Can add communitypolicybundle

Users:

- Amy Zhang
- None
- Grant Hugh

RECENT ACTIONS

- Policyengine | pol
- Policyengine | pol
- Policyengine | pol

QUICK LINKS

Policy filter code:

```
if action.action_type == 'SlackRenameConversation':
    return True
```

Policy init code:

```
usernames = [u.username for u in users]
jury = random.choices(usernames, k=3)
action.data.add('jury', jury)
a
```

abs()
all()
and
any()
apply()
as
assert

Policy notify code:

```
tion.data.get('jury')
rs = users.filter(username__in=jury)
mmunity.notify_action(action, policy, users=jury_users)
```

Policy conditional code:

```
jury = action.data.get('jury')
jury_users = users.filter(username__in=jury)
yes_votes = action.proposal.get_yes_votes(users=jury_users)
if len(yes_votes) == 3:
    return PASSED
elif action.proposal.time_elapsed() > datetime.timedelta(days=2):
    return FAILED
```

Policy action code:

```
action.execute()
```

Policy failure code:

```
pass
```

PolicyKit Website

policy code editor

We have built platform integrations with Slack, Reddit, and Discord and plan to add additional platforms.

You can learn more about PolicyKit and add it to your community at policykit.org

Our code is open-sourced at https://github.com/amyxzhang/policykit

PolicyKit: Building Governance in Online Communities

Authors: Amy X. Zhang, Grant Hugh, Michael S. Bernstein