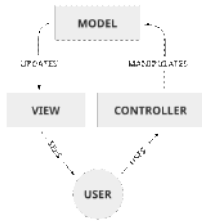


Old Management Style

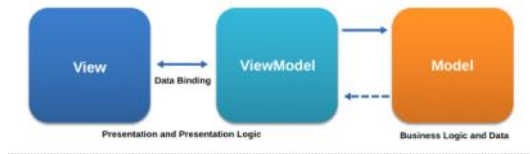
MVC =



Controller links the model and view, the view only changes at the controller's discretion.

MVP is a weak wrapper around MVC.

MVVM =



View no longer depends on the model's certain defined structure.

The ViewModel serves to translate any data to the required view.

MVIT is a view wrapper around MVVM to make event injection more obvious.

VIPER tries to combine all these state architectures, but badly.

Viper forces them together, adding components to complement a lack of understanding, where condensing the presentation would have been more useful and extendable.

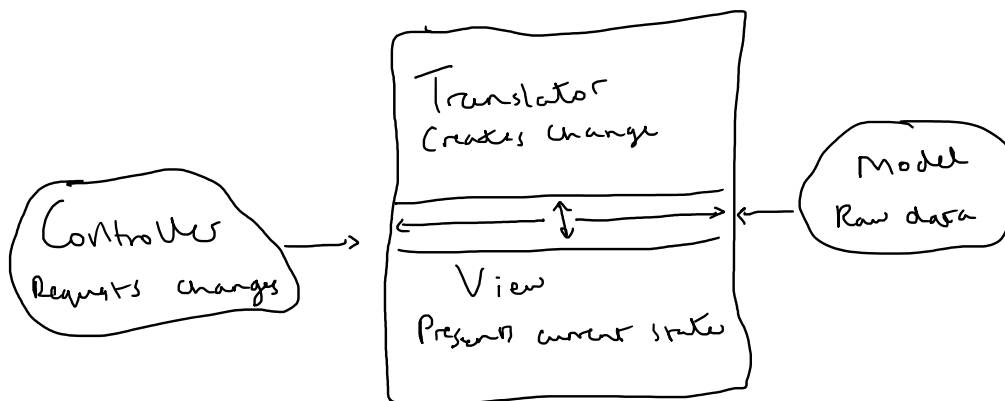
... ..

New: $C[TV]^*M$

An architectural pattern to enable infinite composition with application specific optimization enabled between computation and storage.

Controller - Translator - View - Model.

Note the regex annotation for ' $[TV]^*$ ', to denote both implied isolation from the Controller/Model and to make explicit the infinitely-expansive opportunity by adding any combination of additional translators and views.



Translation and View are coupled but disconnected because infinitely many views can choose which of the infinitely many translations they wish to present.

The controller is no longer tightly coupled with the view or model.

Controllers are no longer guaranteed to be aware of every translation or view, hence only requesting changes and allowing the translators to handle appropriate propagation.

Commands were authoritative and often immediate by design.

A command could previously be like "do this".

A request is more like "x needs to do y after either n minutes or once prerequisite task is complete".

A command is just one small part of a request. The additional data captured by using requests enables much more flexibility in optimal propagation strategies.

Completely disconnecting the model from any controlling authority also allows data to be stored using multiple models where appropriate, based on the translation and views used.

Why $C[TV]^*M$?

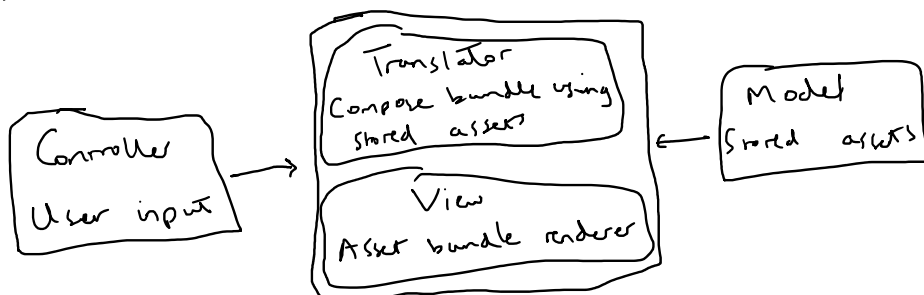
Most archaic patterns are made for a digital world with more dynamic objects. That is: for a digital world where demands have predetermined allowed origins.

Bidirectional communication has been slowly enabled, prompting the iteration attempts. Basic bidirectional comms already warrants use of $C[TV]^*M$, but the simplicity in communication techniques has enabled less informed patterns to prevail.

Gen AI presents an explosion in communication abilities. Direction and number of participants are now both dynamic by design. At this point, a better behaviour pattern is almost a must.

$C[TV]^*M$ in single practice

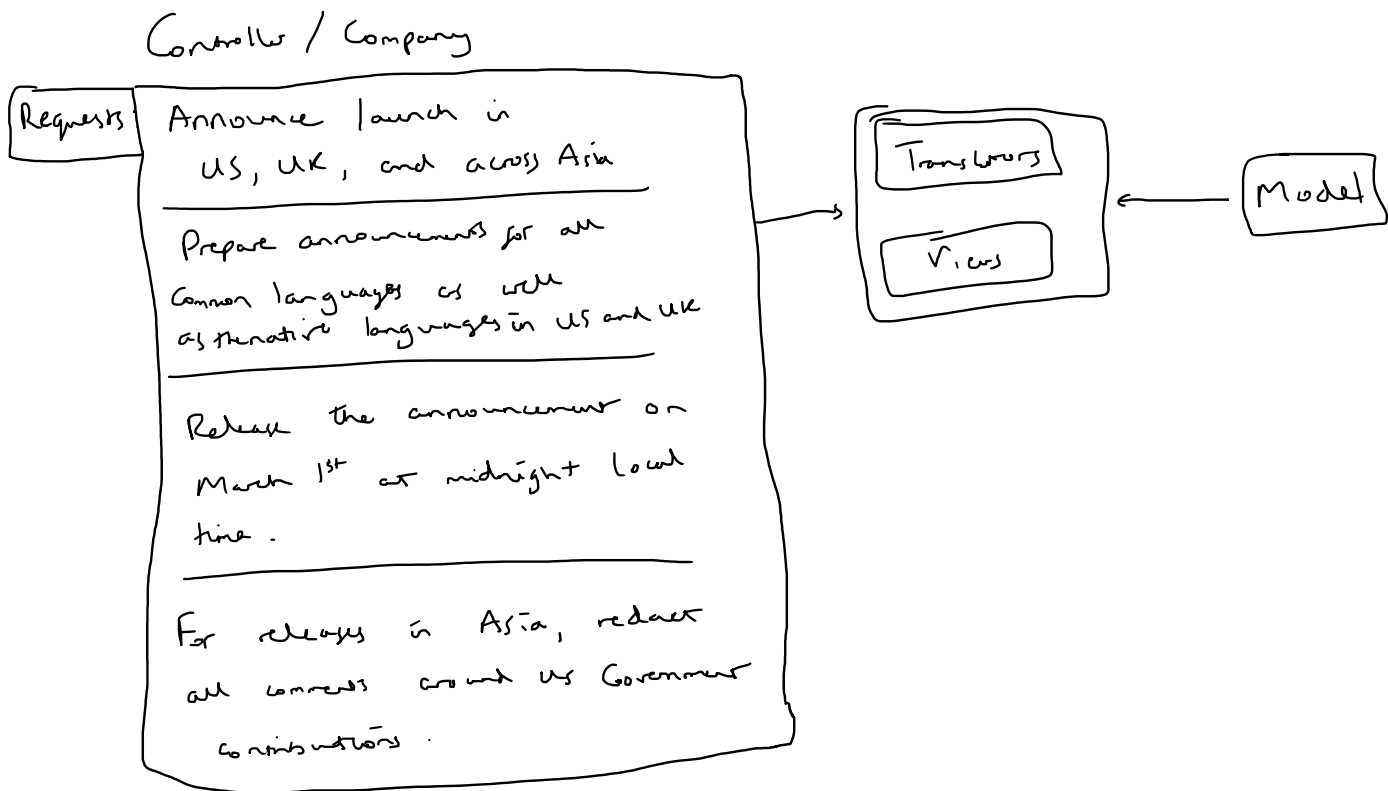
Even the most basic applications are easily expressed using $C[TV]^*M$.



[TV]* allows multiple of either but also implies allowing none of either or both. If the model is the final asset bundle, the view can use it directly without translation. If the user wants to blindly request updates, there is no requirement for any new component. If the controller wants to make exact, authoritative changes they can eliminate translators and views, and update the model directly.

C[TV]*M in real life

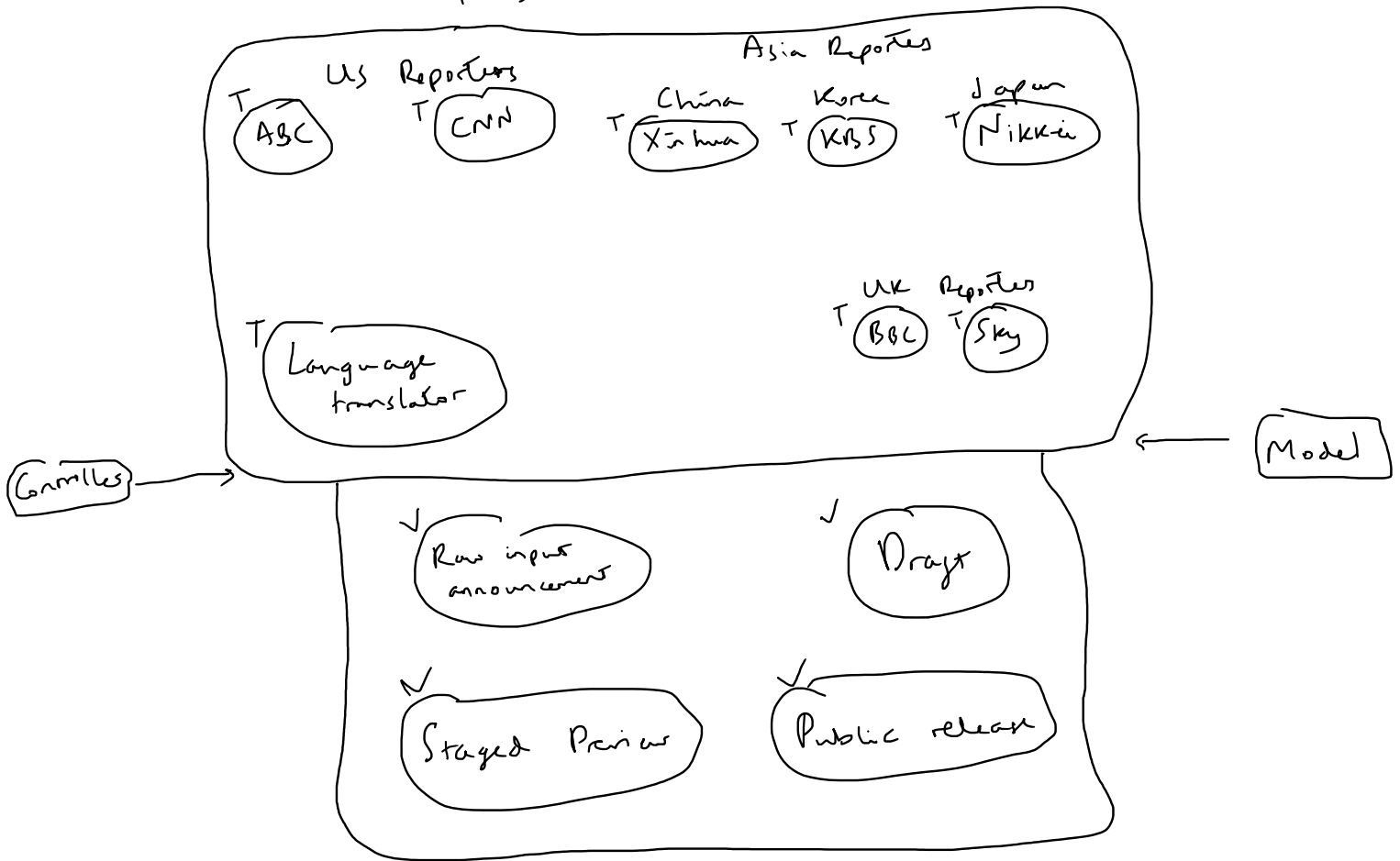
Sharing news across countries is a great example of C[TV]*M in action. Imagine a company has an announcement to make.



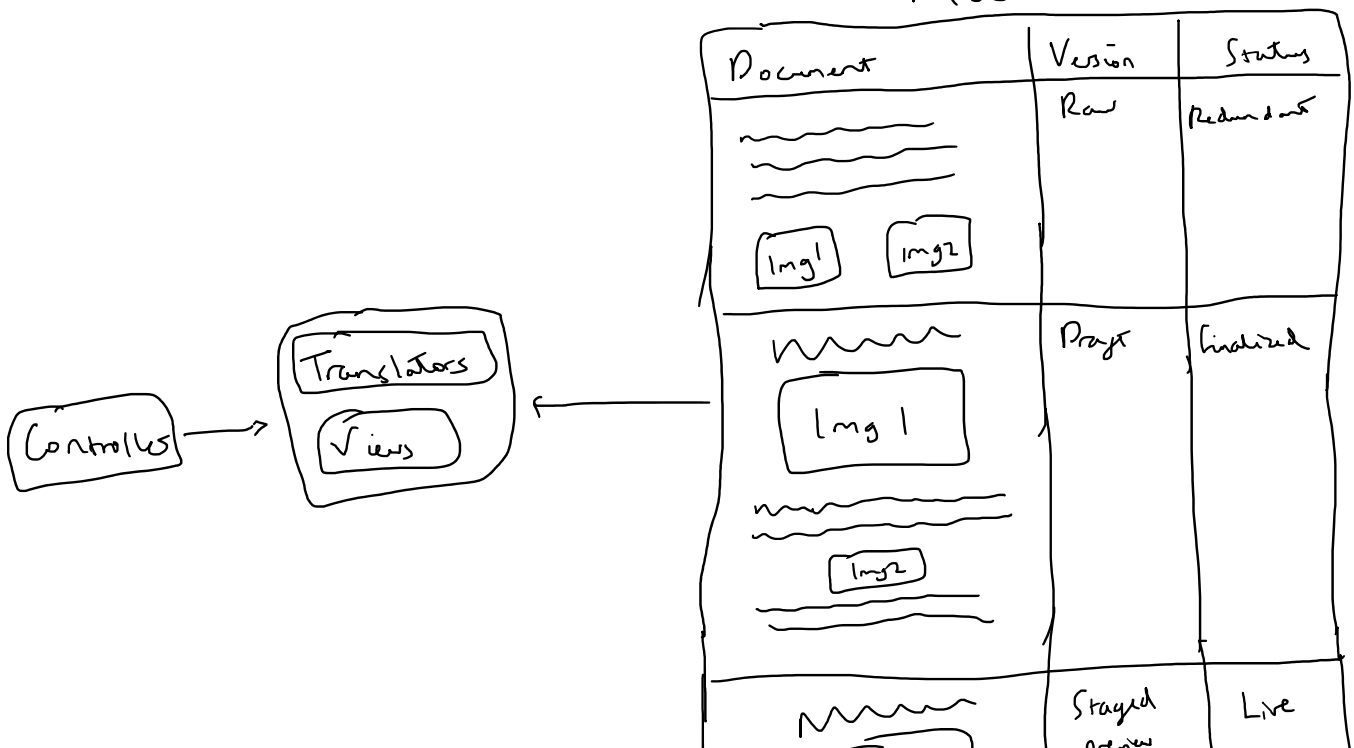
Translators / Views

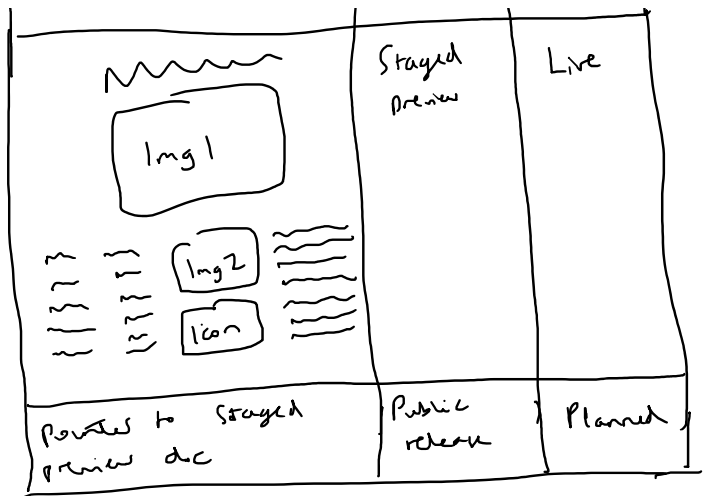
Asia Routes

Translators / Views



Model





While there may be multiple controllers, by default all requests are treated as equal. This keeps multiple-controller systems simple to manage, since all requests are fed through the same interface.

This makes adding requests from new controllers almost trivial.



It is possible that specific input sources will be attached to specific translators during input but that is not a requirement.

For example, government requests might be tied to a translator that adds the detail "for release made in the government's country".

Events are native

An event is no more than a translation. The Controller itself can be considered as an event trigger, where input is just wrapped by a set of default translations.

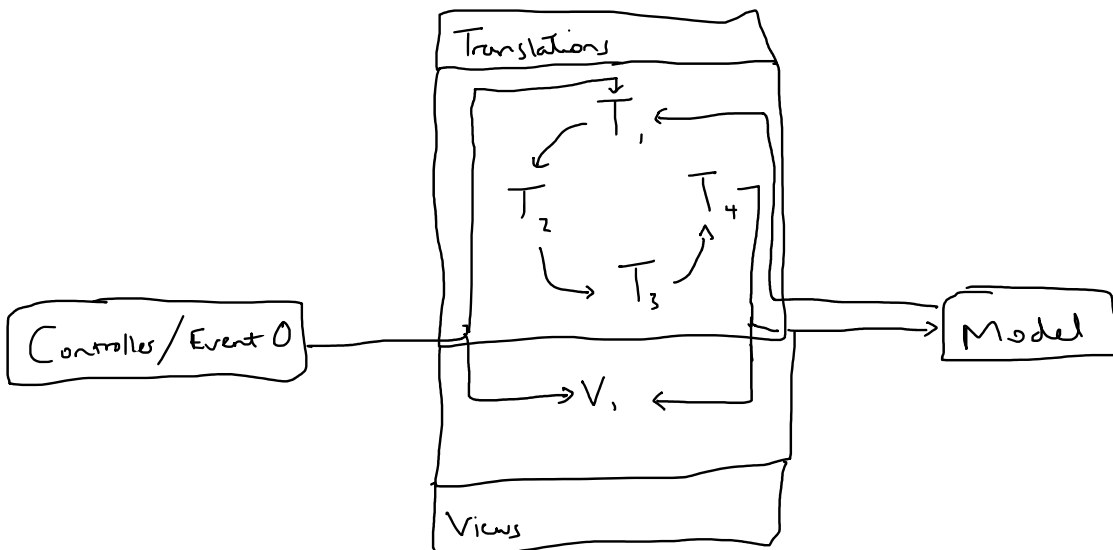
The event origin and purpose determine how it will be wrapped in translations before being considered handled. It may help to consider events as orthogonal to any defined architecture pattern. Events trigger translations and translations can emit events, which is how the chaining becomes infinitely composable.

Translations are created to function as automation. These automation can be reused while grouping them into sets to form any number of base layer finite state machines. Event triggers and subscriptions are mapped to create the first assumed flow state.

Isolated automation reuse enables all external input to undergo purpose specific transformations before being fed into the flowing state machine.

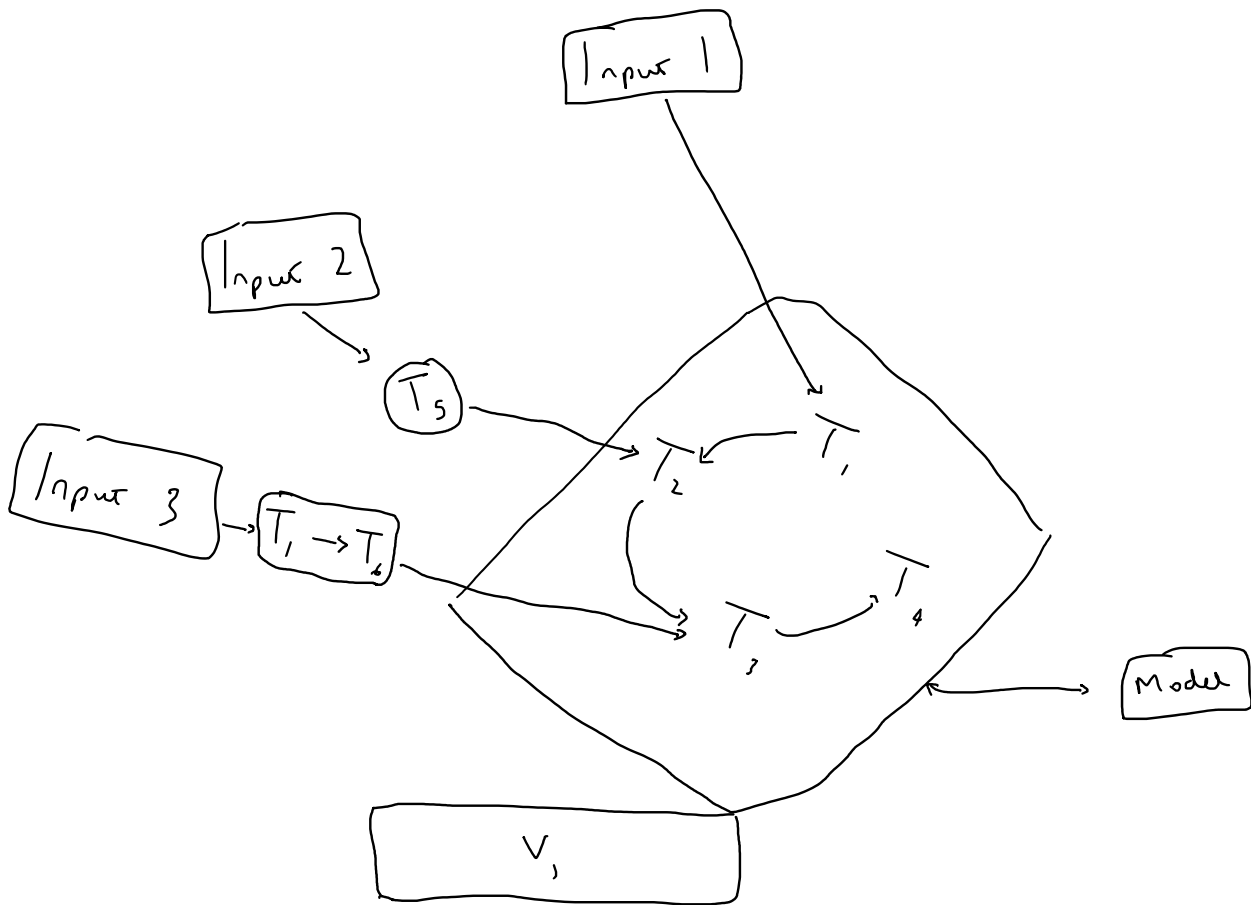
Considering only Controller Events

Model $\rightarrow T_1$ on init
Controller $\rightarrow T_1$ thereafter



v12

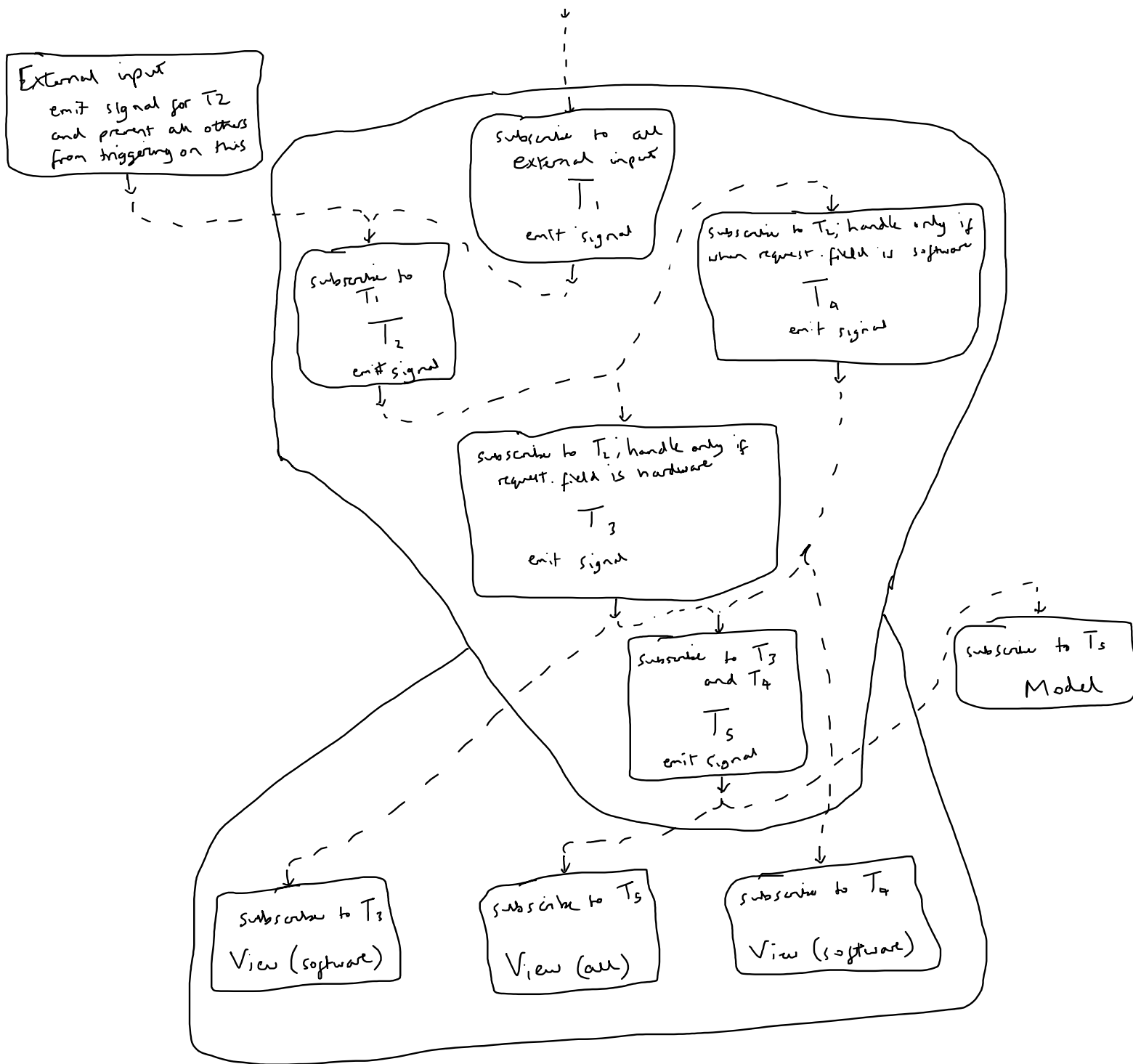
Considering many external events



Why $C[TV]^*M$ is native for Gen AI

Because finite state machines can be fashioned using translations ad hoc,
 $C[TV]^*M$ captures the dynamic signal routing essential for efficient Gen AI.
 Flow via requests instead of comments allows signals to trigger based on the
 state of the request as well as the origin. Equally, signals can end and
 use the request to specify any targets that should be forcefully included or
 excluded, again including conditional targets.

Controller
exit signal



$C[TV]^*M$ is broadly extendable.

While only two concrete examples have been presented, the same architecture pattern can be used to represent all applications, media, and modern communications.

... $C[TV]^*M$ makes analysis

modern communications.

Simple is often better, and the simplicity in $C[TV]^*M$ truly unlocks infinite potential.