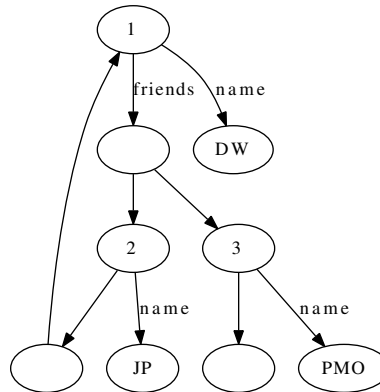
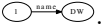



1 Setting: Simplified Facebook

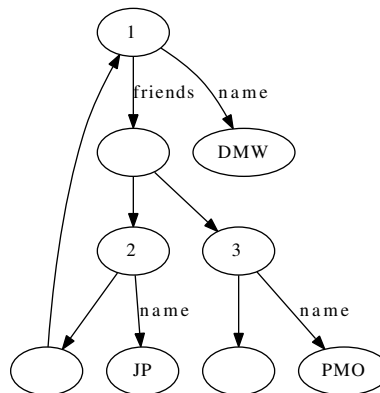
Graphs have three kinds of nodes: user identity nodes, friends list nodes, and name nodes. User identities point to a name node and a friends list node; friends list nodes point to user identity nodes. For example:




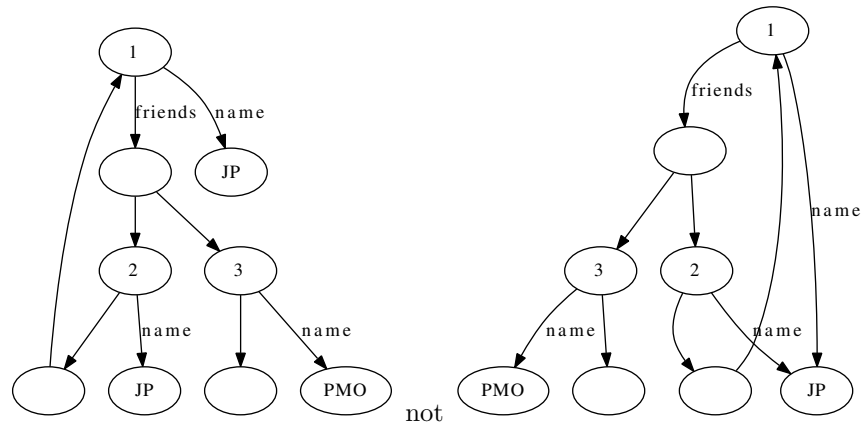
1.1 Choosing the name field

The simplest thing you might want to do is pick out a particular person's name. For example, let's play with a lens that we will call "1:name" for now that picks out the node with user identity 1 and its corresponding name node. In the get direction, the above graph would be in sync with the graph .

We could put back the graph  to get the graph:

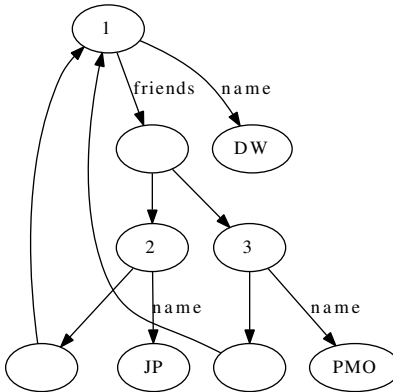


If we put back a graph like  with a name that happens to already be in the graph, we would not want to introduce spurious sharing:



1.2 Choosing the friends field

Let's call this lens "3:friends:*"; it should behave pretty similarly to the last one, picking out PMO's friends. For example, the starting graph would be in sync with the graph . However, unlike the previous lens, if we put back a graph like , we should get some sharing:

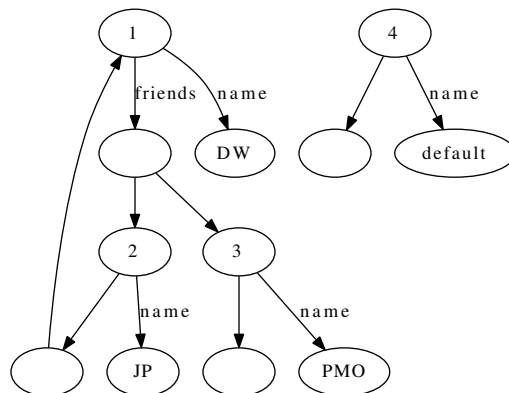


Perhaps either the graph itself or some part of the lens should indicate where sharing is desired and where not. For example, you might imagine having two separate "chunks" of each graph: the chunk where sharing is expected and nodes have permanent node identities, and chunks that are tree-like attached to identified nodes.

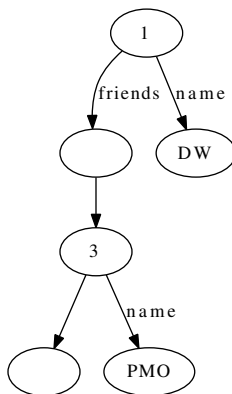
1.3 Creating and deleting users

It's not clear exactly what interface you want for this. You could keep a strict analogy with lenses as we know them. In that case, you would want to be able to write a lens, let's call it "users", which would pick out all the roots of the

graph. For example, the starting graph above would be in sync with $\textcircled{1} \textcircled{2} \textcircled{3}$. You would then create a new user by putting back a graph like $\textcircled{1} \textcircled{2} \textcircled{3} \textcircled{4}$, which would result in something like:



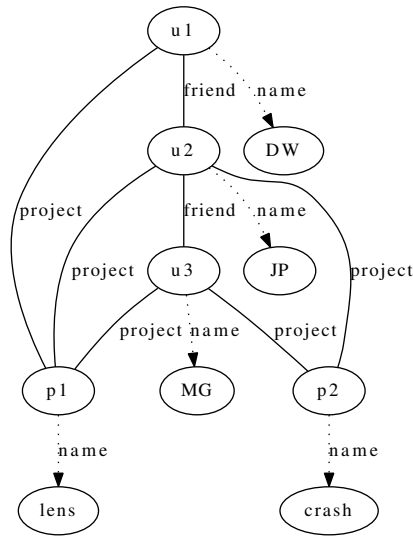
Alternatively, you could put back a graph like $\textcircled{1} \textcircled{3}$ to delete a user and (presumably) garbage collect its fields:



Perhaps one would like some additional operation for creation and deletion, though.

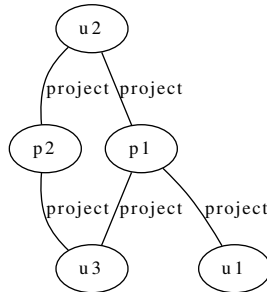
2 Setting: Simplified Github

Graphs have three kinds of nodes: user nodes, project nodes, and names. We will expect the user and project nodes to have identities, and therefore this part of the graph should exhibit sharing during updates, whereas the names should not. Here's a sample graph, using dotted lines for the “unshared” chunks of the graph:

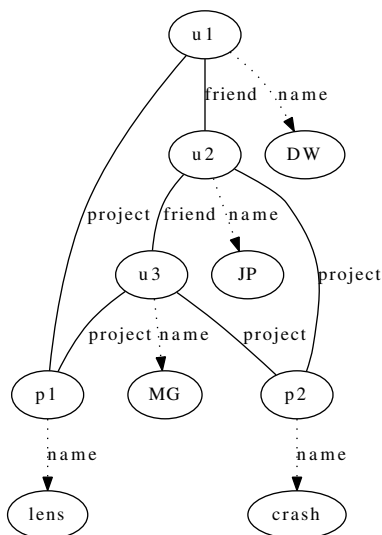


2.1 People who are working together

We might want to pick out all of **JP**'s colleagues. Let's call the lens that does this "u2:project:project". The sample graph above would be in sync with the graph:



We could then indicate that **JP** is no longer working on the crash project by putting back the graph $\text{u2} \xrightarrow{\text{project}} \text{p2} \xrightarrow{\text{project}} \text{u3}$, which would result in:



This graph only differs from the original by the deletion of a project edge between u2 and p1.

2.2 Adjusting names/friendship

As with the simplified Facebook example, we expect that changing names doesn't produce sharing but changing local friendship subgraphs might; so we distinguish between these subgraphs using the solid and dotted edges.

2.3 Creating or deleting users and projects

The discussion of the previous section applies almost verbatim: we could try to cast this as a classical lens by creating one which picks out just the user nodes or just the project nodes, but even after doing that it may make sense to have a separate operation for creation (since, after all, it should probably be the system's job to pick a globally unique ID for the new user or project).